NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

DESIGN AND IMPLEMENTATION OF
AN MC68020-BASED
EDUCATIONAL COMPUTER BOARD

by

Yavuz Tugcu

December, 1989

Thesis Advisor: Gerald J. Lipovski

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
DESIGN AND IMPLEMENTATION OF AN MC68020-BASED EDUCATIONAL COMPUTER BOARD

12 PERSONAL AUTHOR(S)
TUGCU, Yavuz

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1989 December	135

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the US Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	microprocessor; coprocessor; Microprocessor Development System

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The goal of this thesis is to design and implement a Motorola 68020 based Educational Computer Board (ECB), including the Motorola 68881 co-processor. The ECB has two communication channels, one for an external I/O device and the other for a Macintosh personal computer. A stored program can be installed in 8K bytes Programmable Read Only Memory (PROM) to initialize the ECB and to handle communication, as well as to perform user commands via a Macintosh personal computer.

The ECB operates at a clock frequency of 16 MHz. It includes four Static Random Access Memory (SRAM) chips which provide a storage of 32K bytes. Two Programmable Array Logic (PAL) chips generate the required decoding, enabling and timing signals. No special I/O chip is used in Macintish interface, except for a RS-232 line driver/level changer, as the

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL LIPOVSKI, Gerald J.	22b TELEPHONE (Include Area Code) 512-471-1952 22c OFFICE SYMBOL 52

DD Form 1473, JUN 86 *Previous editions are obsolete*

19. continued

communication on this channel is intended to be under software control in order to keep the hardware as simple as possible. The channel for an external device has not been implemented and tested, but all the required pads and holes are available to install 74244 and 74245 TTL line driver IC's for this channel.

Approved for public release; distribution is unlimited.

Design and Implementation
of
an MC68020-Based Educational Computer Board

by

Yavuz Tugcu
1st Lieutenant, Turkish Air Force
B.S.E.E., Middle East Technical University, Ankara, 1981

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

Department of Electrical and Computer Engineering

ABSTRACT

The goal of this thesis is to design and implement a Motorola 68020-based Educational Computer Board (ECB), including the Motorola 68881 coprocessor. The ECB has two communication channels, one for an external I/O device and the other for a Macintosh personal computer. A stored program can be installed in 8K bytes Programmable Read Only Memory (PROM) to initialize the ECB and to handle communication, as well as to perform user commands via a Macintosh personal computer.

The ECB operates at a clock frequency of 16 MHz. It includes four Static Random Access Memory (SRAM) chips which provide a storage of 32K bytes. Two Programmable Array Logic (PAL) chips generate the required decoding, enabling and timing signals. No special I/O chip is used in Macintosh interface, except for a RS-232 line driver/level changer, as the communication on this channel is intended to be under software control in order to keep the hardware as simple as possible. The channel for an external device has not been implemented and tested, but all the required pads and holes are available to install 74244 and 74245 TTL line driver IC's for this channel.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Microprocessors continue to be an integral part of many complex digital systems. Through improvements in manufacturing techniques, they have become more powerful and more complex. This power and flexibility is accompanied by increased complexity and difficulty in hardware and software design. The hardware designer must consider more control and data signals. Similarly software design entails more detailed considerations. The complexity of a microprocessor-based system also increases the difficulty of maintenance and troubleshooting. The operation of such a system should be throughly understood before attempting any troubleshooting action.

The manufacturers of microprocessors have introduced new products so often that the number of people who know and use these products is somewhat limited. The best way of learning a system is through using it. This idea forms the basis for the thesis presented here. Within the scope of the thesis, an Educational Computer Board (ECB) has been designed and implemented to be used

- as a tool for teaching a state-of-the-art microprocessor and coprocessor design, and
- as an experimental, test, or control device for scientific applications.

In the design of ECB, the main consideration was to use the minimum number of external components to achieve simplicity, low-cost and reliability.

In the chapter "An overview of MC68020 and MC68881", the basic operations of main processor and co-processor are reviewed. The chapter "Design and Implementation of the ECB" discusses several design alternatives and explains why a particular design has been selected. The chapter "Hardware verification" includes the outputs of a series tests to verify the operation of the ECB. A comparison is made, in the last chapter "Conclusions", between the ECB developed in this thesis and the ECB previously designed by Motorola and still in use in microprocessor-based courses at the Naval Postgraduate School. Also suggested future improvements are given in this last chapter.

II. AN OVERVIEW OF MC68020 AND MC68881

This chapter introduces the architecture and features of the MC68020 and its associated coprocessor MC68881. Also given is a brief description of the signals and the interface between two processors. Detailed information on the signal description, timing and instruction set is given in **Appendix A, B** and **D**.

A. MC68020 Architecture and Features

Implemented in VLSI technology, the MC68020 is upwardly compatible with its predecessors, the M68000 and M68010. That is, in addition to the new instructions, all the instructions that run on the old M68000 family members, can be run on MC68020. All I/O devices that can be connected to the M68000 and M68010 can also be connected to MC68020. A table of MC68020 instructions and new instructions which are extensions to old M68000 family members are given in **Appendix D**.

The MC68020 has an 128 word on-chip cache memory (compared to 3-word cache memory in M68010 and no cache memory in M68000). The advantage of cache memory is to reduce both the total execution time of a program and the external bus activity of the processor without degrading the performance. The basic idea is to store the instruction stream prefetched from main memory into the faster on-chip cache memory so that the processor does not have to access main memory to fetch the next instruction in most cases. This on-chip cache memory can be enabled or disabled by applying an external signal to the chip. The ECB has been implemented with this feature disabled.

The MC68020 contains 32-bit data/address registers and 32-bit data/address buses. Thus, it can directly address a memory range of 4 Gigabytes. In each bus cycle, the microprocessor can determine the port size of the external device to or from which an operand is to be transferred. This feature is called "Dynamic bus sizing". The MC68020 can be connected to external devices having port sizes of 8, 16 or 32-bits, so all data alignment restrictions are eliminated. On the ECB, 32K byte ROM is connected as an 8-bit port and 32K byte RAM is connected as a 32-bit port. An input signal can inform the microprocessor, if an external device does not respond to a command within a specified period of time, so that the microprocessor can initiate a new bus cycle. This

signal (BERR) which is to be generated by an external circuit has not been used on the ECB, as the purpose in the design is to use minimum hardware to the degree that guarantees the proper operation of the ECB, as well as to make it easy for the programmer to write the software that will handle the operation of the ECB, during this development phase. **Appendix C** includes the information on "Bus Operation" and focuses on dynamic bus sizing and multiplexing of the data onto the external bus.

The MC68020 has three processing states, and it is always in one of these states: normal, exception and halt. In the normal state, the processor executes instructions (fetching instructions and operands, storing results and communicating with the co-processor). If an unusual condition (exception) occurs during normal instruction execution, the processor enters the exception state to handle this condition easily. An exception can be generated internally by an instruction or externally by an interrupt, reset, etc. The processor enters the halt state whenever it detects a system failure. In halt state, there will be no processor activity, until an external reset (the only means to regain the processor activity) is applied to restart the processor. The halt state is not the same state as the stopped state which is caused by STOP instruction. The instruction execution on a stopped processor resumes after a trace, interrupt or reset exception.

Within each of the three processing states, there are two privilege levels, user and supervisor. The supervisor state has higher privilege than the user state, so that all processor instructions are available to execute in this state. In the user state, programs are allowed to access only their code and data areas, and they cannot execute some processor instructions related to system functions. This provides security in the microprocessor system.

The MC68020 behaves slightly differently in the supervisor state than the old M68000 family members. It allows the separation of supervisor stack space for user tasks and for interrupt associated tasks in order to increase the efficiency in a multi-tasking operaing system. This separation is enabled by setting the M bit in the status register. The M bit is cleared, whenever an exception occurs for interrupts. The processor can be switched from the user state to the supervisor state only through exception processing. Switching from the supervisor state to the user state is accomplished by executing an instruction that can modify the status register. Figure 1 shows the positions of the status and control bits in the status register.

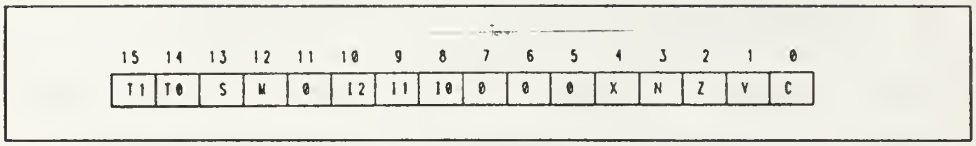


Figure 1 Status Register

The MC68020 has three defined types of address space, encoded by the function code pins FC0-FC2. These address spaces are the user data/program space, the supervisor data/program space and the CPU space, as shown in Table 1.

Table 1 MC68020 Address Spaces.

FC2	FC2	FC0	ADDRESS SPACE
0	0	0	UNDEFINED
0	0	1	USER DATA SPACE
0	0	0	USER PROGRAM SPACE
0	0	1	UNDEFINED
1	0	0	UNDEFINED
1	0	1	SUPERVISOR DATA SPACE
1	1	0	SUPERVISOR PROGRAM SPACE
1	1	1	CPU SPACE

The user and supervisor address spaces have no predefined memory locations, except for the addresses of the initial interrupt stack pointer and program counter values that are held in the first two longwords of the supervisor program space. The MC68020 fetches these two longwords and loads them into the interrupt stack pointer and the program counter, respectively, by reading from supervisor program space. CPU space accesses are made when the processor communicates with the external devices for data movements other than those associated with instructions, like interrupt acknowledgements and coprocessor operation. During CPU space accesses, address lines A19 through A16 specify the type of CPU space, as shown in Figure 2.

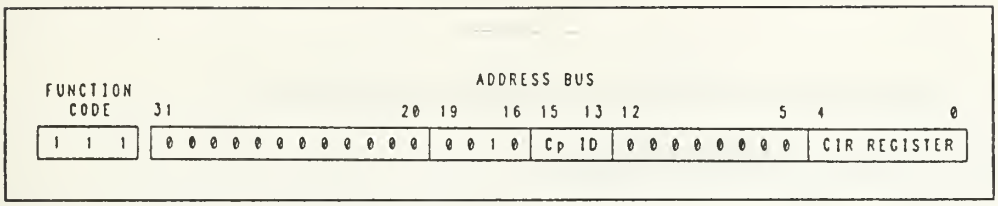


Figure 2 CPU Space Encoding

On the ECB, the address lines A18, A17 and A15 are used to generate the chip enable signal for the MC68881 coprocessor. The function code lines FC0-FC2 are not incorporated in the co-processor chip select generation circuit.

In the processing of an exception, the MC68020 goes through four identifiable steps.

1. An internal copy of the status register is saved temporarily and the status register is set to process the exception.

2. The exception vector is generated. An exception vector is a pointer to the memory location containing the address of the routine which handles the specified exception. There are 254 exception vectors available in the supervisor data space, and 2 vectors for the reset exception in supervisor program space. A group of 64 vectors is defined by the processor and the remaining 192 vectors are left for user to define. Exception vectors can be generated externally or internally. On the ECB, all the interrupts are auto-vectored, that is, the exception vectors are generated internally by the processor upon the recognition of the interrupt.

3. The current processor context is saved on the exception stack frame created on the active supervisor stack. This context always includes the status register, the program counter and the vector offset for the exception vector. Another field on the exception stack frame called "format field" is used to specify what additional processor state information has been put onto the stack frame, as there is more than one type of exception stack frame created by different exceptions.

4. At the last step, the address of the exception handler is loaded into the program counter, then the instruction at that address is fetched and the program execution is

resumed.

For detailed explanation on exception processing, see **Appendix D**.

B. MC68881 Architecture and Features

The MC68881 floating-point coprocessor is implemented in VLSI and HCMOS technology which combines the HMOS (High Density NMOS) and CMOS technologies to achieve low power, high speed and minimum silicon area. Although it is primarily designed for use with MC68020 microprocessor, it can also be used with the old M68000 family members with some degradation in the performance. This is due to the fact that the MC68881 is recognized as a coprocessor by the MC68020 and as a peripheral processor by the other M68000 family members. The data bus on which MC68881 operates can be 8, 16 or 32-bits wide. The MC68881 has eight 16-bit and four 32-bit co-processor Interface Registers (CIR) which are memory-mapped to the CPU address space of MC68020 in order to provide exchange of commands and data.

From the programmer's point of view, the pair MC68020/MC68881 can be thought as one MC68020 processor implemented on the same chip, having additional eight floating-point data registers. Each floating-point data register is 80-bit wide (1 sign bit, 64 bits for mantissa and 15 bits for exponent). The MC68881 fully conforms to IEEE P754 Binary Floating Point Arithmetic Standard and supports seven data types: byte, word, long integer, single, double, extended precision real and packed BCD real. There are 22 scientific constants available on the chip.

Appendix E includes detailed information on the MC68881 registers and data types. **Appendix B** contains the MC68881 signal description.

C. The Interface Between the MC68020 and MC68881

The interface between the MC68881 and the main processor is provided by the M68000 Family coprocessor interface which allows connection of up to eight co-processors. Each co-processor is addressed by driving its ID number on the address lines A13 through A15. On the ECB, these lines are not decoded to generate the chip

select signal, as there is only one co-processor which is always addresssed for any ID number.

The main processor MC68020 communicates with the floating point coprocessor MC68881 over a 32-bit data bus, and accesses the coprocessor interface registers through bus cycles. Each interface register (CIR) has a specific function and is used as a communication port. The coprocessor connection diagram for 32-bit data bus is given in **Appendix F**. On the ECB, function codes FC0-FC2 are not used for the generation of chip select signal.

The interface tasks are divided between the MC68020 and MC68881 so that they do not duplicate each other's functions. For example, the main processor does not have to decode the co-processor instructions; it is the responsibility of the co-processor to decode these instructions. On the other hand, the coprocessor does not involve the calculation of the effective address. It only instructs the main processor to transfer an operand over the interface, then it is the responsibility of the main processor to calculate the effective address and fetch the operand. Thus, the coprocessor never becomes a bus master.

III. DESIGN AND IMPLEMENTATION OF THE ECB

This chapter gives a brief description of how the ECB has been configured and discusses the design of the external hardware, as well as the benefits of the particular design selected. The description of external circuits are not given in detail in this chapter. **Appendix G** can be referred to for detailed information.

A. Introduction to the Design

Before going into the details of the design, the configuration of the ECB had to be determined, that is, what external devices would be connected to the main processor and in which way they would be connected. The ECB was intended to communicate with a smart terminal to download user programs and to issue commands for running the downloaded programs and for manipulating the other ECB functions. Thus, the first external device was a smart terminal, like a personal computer. Memory was the second external device to exist on the ECB, since every processor needs some memory for storing programs and data. The last external device was the MC68881 coprocessor.

Once the configuration of the ECB had been determined, the next step was to design the external circuits which would provide the required interface between the MC68020 and the three external devices. The main objective in the design of the external circuits was to keep the hardware at the required minimum to allow proper operation of the ECB in the simplest and primitive way.

All handshake signals for three external devices are generated by two Programmable Array Logic IC's PAL16R4 and PAL16L8. Another integrated circuit, MAX232 converts the RS-232 line voltage levels to TTL-voltage levels and vice versa, with a +5V power supply only. Except for the reset and software abort circuits, the other main components on the ECB are the MC68020 processor, the MC68881 coprocessor and the memory chips (1 27C256 ROM chip, 4 6164 RAM chips).

In the following sections the interface with each external device will be discussed separately.

B. Interfacing with the Memory

The memory to be implemented should be large enough to hold the initialization and user programs/data, as well as the basic routines, but small enough to keep the hardware simple and inexpensive. Small memory size also allows the high order address bits to be used for other purposes, like RS-232 transmission and reception, as explained later.

1. Non-volatile memory (ROM) is used to hold the initialization data and routine during power-up. An 8K byte ROM is sufficient for that purpose.

2. Volatile memory (RAM) is required to hold the user programs and data to be downloaded via the RS-232 interface. A 32K byte RAM was found to be satisfactory for this.

An important design consideration is what kind of information is to be stored in non-volatile and volatile memories. It was decided that low level routines for initialization, I/O (input/output) and exception handlers would be kept in ROM in order to provide security for basic routines which should not be destroyed by overwriting. A requirement imposed by the system is that ROM must be accessed in the very first addresses to allow for fetching the initial interrupt stack pointer and program counter values (reset exception vector). On the other hand, it is very convenient for the user to write his/her own exception routines and to change the contents of the exception vector table located in low memory. This stipulates that both ROM and RAM must be mapped into low addresses which is the case with the current implementation during power-up or reset. In order to prevent collision on the data bus, all reads are made from the ROM and all writes go to RAM when both memories are mapped into the same space. This allows both to fetch the reset exception vector and to copy the contents of ROM to RAM. After a copy of ROM is made to RAM, then the ROM is detached from the low addresses and mapped into higher addresses. Only RAM is accessible for reading and writing in the low addresses thereafter. By this scheme, the user can change the exception vectors in low memory RAM and can access the basic routines in high memory ROM.

The last step in the design process was the development of the external circuits in order to generate the required interface signals with proper timing. These signals are generated by two PAL (Programmable Array Logic) circuits. PAL A (PAL16R4) generates the signal (PHANTOM) which detaches ROM from low addresses and maps into higher addresses. PAL A also returns "Data size and acknowledge" signals (DSACK0, DSACK1) which tells MC68020 that an 8-bit port (or 32-bit port) has been accessed when ROM (or RAM) was addressed. PAL B (PAL16L8) generates the chip select signals for both ROM and RAM, and Read/Write signal for RAM, thus performing the memory mapping. DSACK signals for ROM accesses are delayed to make sure that correct data has been put on the data bus, since the ROM chip has a longer access time than the RAM chips. The volatile memory has been chosen for static RAM (SRAM) which eliminates the refresh hardware required for Dynamic RAMs and provides faster access time.

As a summary of memory interfacing: the size of the memory (8K byte ROM and 32K byte RAM) is sufficient for most programs and leaves high order address bits to be used for other purposes. Static RAM helps the designer reduce the hardware. It also provides fast access and reliability. The memory mapping scheme imposes access of ROM in low addresses, during power-up or reset, and in high addresses after initialization, in order to execute the basic routines. This technique enables the user to modify the system data located in RAM in the low address region. **Appendix G** covers more detailed information on memory interfacing.

C. Interfacing with a Smart Terminal

The ECB communicates with a smart terminal via a serial RS-232 interface. The serial interface is simple; it requires only three wires, but it is slower when compared to a parallel interface with the same clock rate. A voltage level converter chip matches the signal levels on the ECB to the RS-232 line. No special I/O (input/output) chip has been used. The reception and transmission on the RS-232 interface has been implemented in software to keep the hardware simple (see Reference 1). Input and output signals for RS-232 are passed through PAL A and buffered by the level converter chip. Setting the address lines A19 and A15 to high causes a zero to be transmitted on the RS-232 line. On the other hand, setting the address lines A19 and A17 to high causes the RS-232 line to be strobed. If the line is found high (a zero bit)

then an autovectored interrupt of level 4 is generated. The reception of the incoming byte can be handled by the interrupt handler pointed by level four interrupt vector entry. With this scheme, the communication with a smart terminal is only possible when RS-232 line is monitored by the software on the ECB. For detailed explanation and circuit diagrams of this interface see **Appendix G, H, and I.**

D. Interfacing with MC68881 Coprocessor

The most efficient and fastest interface between the MC68020 and its dedicated coprocessor MC68881 is via a 32-bit data bus. Both processors use the same clock, although they can run on different clocks. The connection of most signals are straightforward and direct. The only signal to be dealt with here is the chip select (CopE) which is generated by PAL B out of the address lines A18, A17 and A15. The chip select signal for the coprocessor is also used in the generation of another signal (PHANTOM) which detaches ROM from low address region after initialization. **Appendix G, H and I** include the detailed explanation of the interface and the generation of the chip select signal.

E. Reset and Software Abort Circuits

The main processor and the coprocessor must be reset in order to set their states and registers to predefined and known values. This arises in two cases, initial power-up and reset after a catastrophic system failure in order to bring the system up. It is guaranteed that both processors recognize the reset condition if their reset inputs are held low at least 100 ms by the external circuit. The reset circuit which has been built around Motorola's low voltage detector is quite simple. An external resistor-capacitor combination provides the required delay of at least 100 ms.

In case the user program runs out of control or enters an unintended loop for any reason, the user must have a means to abort the program and return to a defined point before re-running the program without resetting the processor. This is provided by the software abort circuit consisting of all passive components. The circuit generates an autovectored interrupt of level 6, upon pushing the software abort button. The level of the interrupt is one less than the non-maskable highest level. The reason for choosing a level 6 interrupt rather than a level seven interrupt is that the output of the abort circuit

is not debounced. This causes more than one interrupt to occur sequentially, after the software abort switch is released. If a level seven interrupt is generated by the software abort circuit, all the successive interrupts (non-maskable) due to non-debounced output will be recognized. This imposes a delay in the processing of the interrupt, and unnecessary pushes onto the stack, until the bouncing of the switch stops. Assigning a level 6 interrupt to the software abort function improves the response considerably. In the interrupt handler for the software abort, the mask level in the status register is set to 7, before beginning the exception processing so that further level 6 interrupts are not recognized (see Reference 1). This greatly reduces the number of spurious level 6 interrupts that are recognized after the first one.

IV. HARDWARE VERIFICATION

After implementation, the hardware has been verified by running a series of short routines to test the following:

- ROM read.
- Generation of the coprocessor chip enable **CopE** and **PHANTOM** signals.
- RAM read/write.
- Coprocessor communication.
- Interrupt 4 (RS-232 reception) operation.
- Interrupt 6 (Software Abort) operation.

All the tests have been conducted by using the debugger in Reference 1 and the Logic Analyzer HP1650A.

A. ROM Read Test.

The routine for the ROM read test is the RS232 reception routine, itself, which resides in the ROM (See Reference 1). A part of the state listing for this routine is given in Figure 3, in which the MC68020 makes sequential reads from supervisor program space. **DSACK** signals generated by the PAL B return an 8-bit port size for the ROM. The timing waveforms are shown in Figure 4, where it can easily be seen that function codes (**FC2** through **FC0**) are encoded for supervisor program address space. **!DSACK1** signal stays high all the time and only **!DASCK0** is asserted, after **!DS** and **!AS** are asserted, to indicate an 8-bit port size. The ROM chip enable signal **ROMCE** is the only chip select signal that is active. Figure 6 shows the relation between **!DSACK0** and **!AS, !DS** in an expanded scale. The X marker is at the point where **!AS** and **!DS** are asserted, and the O marker is at the point where **!DSACK0** is asserted. The specification for the time between two markers is 80 ns maximum (See **Appendix A**). The measured time is 70 ns as seen in Figure 5.

68020 - State Listing

Markers: Off

Label >	ADDR	DATA	STAT	DSACK
Base >	Hex	Hex	Symbol	Symbol
+0000	000404DA	4E000000	SUPR PGRM READ	8 BIT PORT
+0001	000404DB	F9000000	SUPR PGRM READ	8 BIT PORT
+0002	000404DC	00000000	SUPR PGRM READ	8 BIT PORT
+0003	000404DD	0E000000	SUPR PGRM READ	8 BIT PORT
+0004	000404DE	04000000	SUPR PGRM READ	8 BIT PORT
+0005	000404DF	E0000000	SUPR PGRM READ	8 BIT PORT
+0006	000E04E0	4E000000	SUPR PGRM READ	8 BIT PORT
+0007	000E04E1	71000000	SUPR PGRM READ	8 BIT PORT
+0008	000E04E2	4E000000	SUPR PGRM READ	8 BIT PORT
+0009	000E04E3	71000000	SUPR PGRM READ	8 BIT PORT
+0010	000E04E4	4E000000	SUPR PGRM READ	8 BIT PORT
+0011	000E04E5	71000000	SUPR PGRM READ	8 BIT PORT
+0012	000E04E6	4E000000	SUPR PGRM READ	8 BIT PORT
+0013	000E04E7	F9000000	SUPR PGRM READ	8 BIT PORT
+0014	000E04E8	00000000	SUPR PGRM READ	8 BIT PORT

Figure 3 State listing for the ROM read test

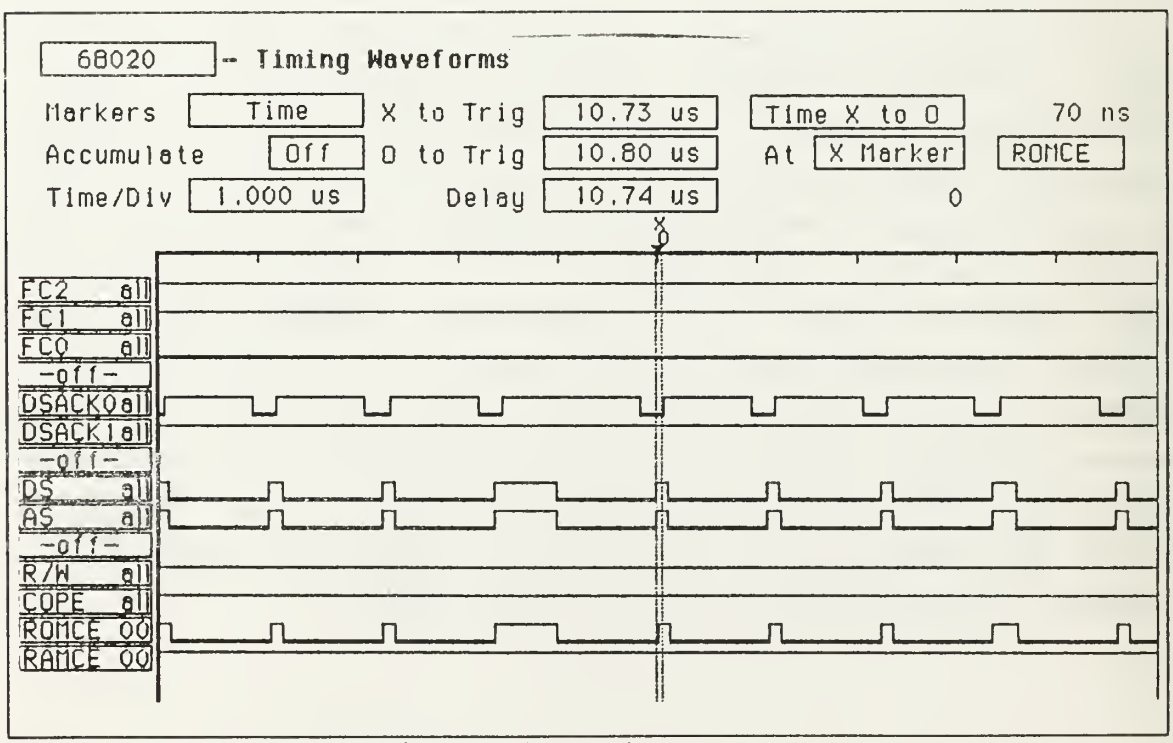


Figure 4 Timing waveforms for the ROM read test

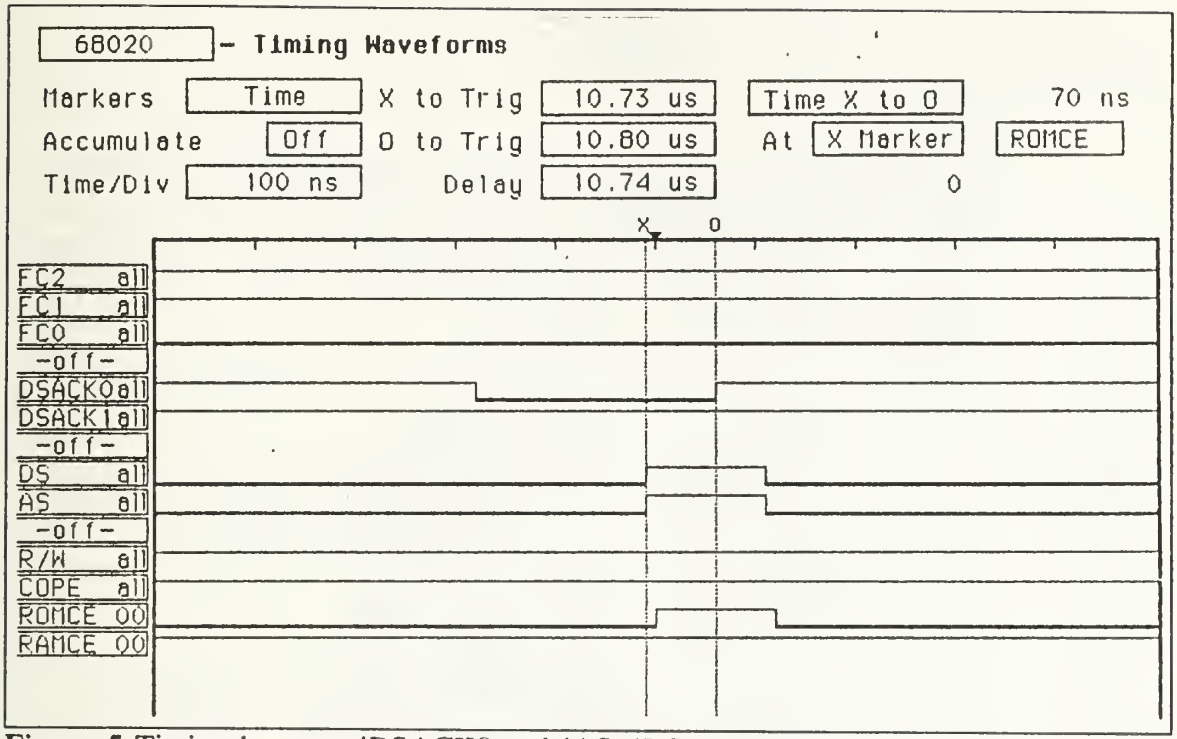


Figure 5 Timing between !DSACK0 and !AS, !DS

B. Testing the Coprocessor Enable (CopE) and Phantom signals.

The test routine for these two signals is the initialization routine for the ECB (See Reference 1). The state listing for part of the routine is shown in Figure 6. When the **PHANTOM** signal is high (default state after a reset or power up), an image of the ROM is mapped onto the RAM, and all reads are made from ROM, whereas all writes go to the RAM. After the **PHANTOM** is driven low, the ROM image is removed from RAM region and the RAM can be accessed for both reading and writing. The only way to drive the **PHANTOM** low is to make a coprocessor access. In the initialization routine, the coprocessor is accessed by **MOVE.L** instruction to read data from $20000, which is shown as supervisor data space in Figure 6 (lines +0000 and +0001).

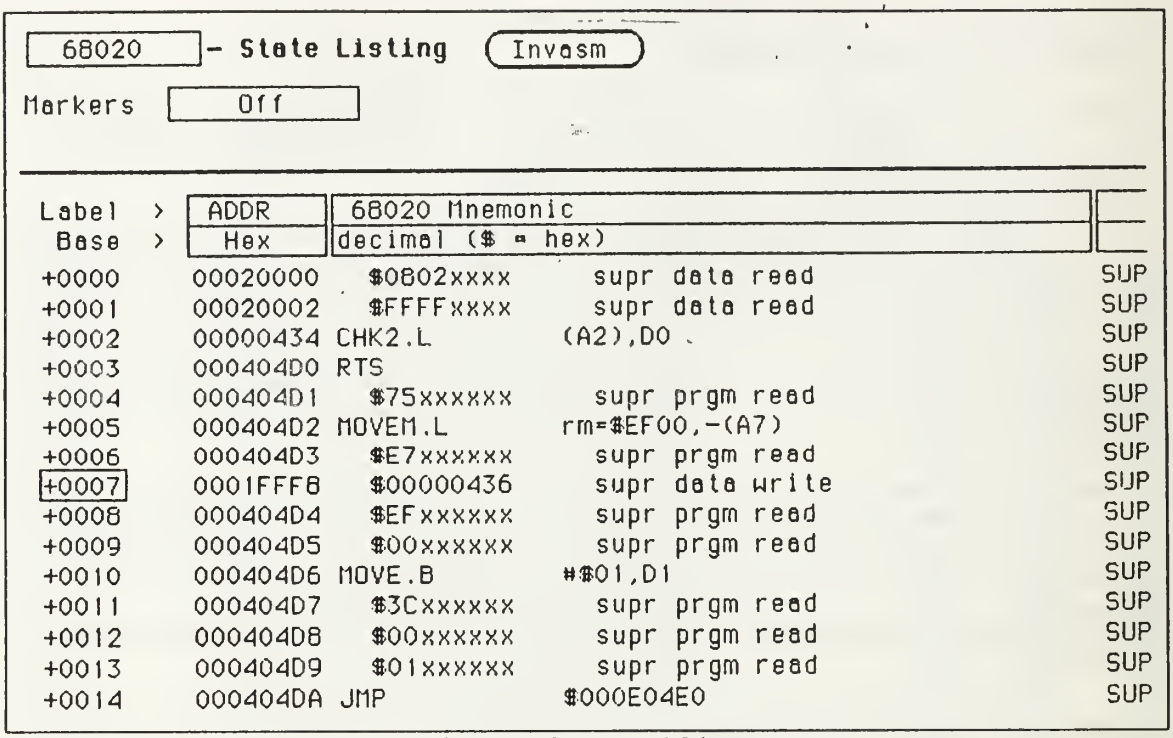


Figure 6 State listing of the routine for CopE and Phantom tests.

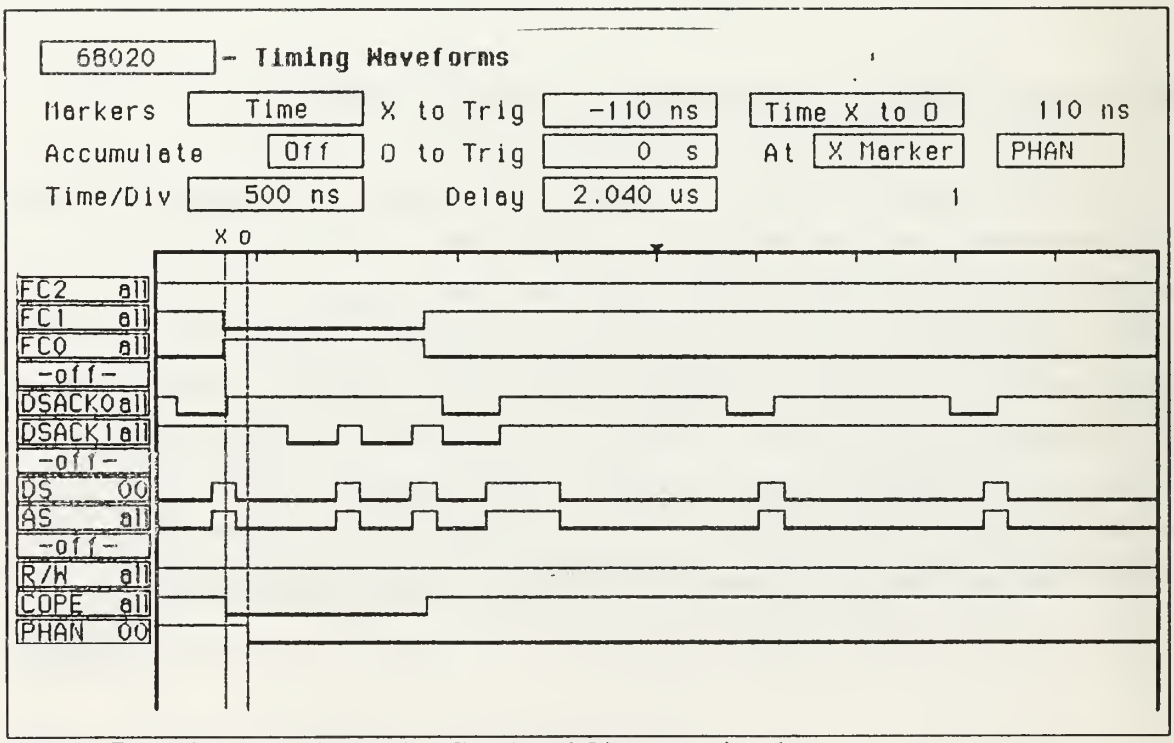


Figure 7 Timing waveforms for CopE and Phantom signals.

Figure 7 shows the timing diagram for **CopE** and **PHANTOM** signals. The **PHANTOM** signal goes low 110 ns after the **CopE** signal is asserted, and it is not affected by the negation of **CopE**. The first read operation after the negation of **CopE** is made from RAM, which is only possible when the **PHANTOM** is low. (See line +0002 in Figure 7 and the point where both **DSACK** signals are driven low simultaneously, to indicate 32 bit RAM port, in Figure 7).

C. RAM Read/Write Test

The routine for this test was downloaded by using the debugger in Reference 1. Figure 8 shows the piece of the code.

```
00001000    21FC555555556000 MOVE.L   #1431655765,$00006000
00001008    20386000         MOVE.L   $00006000,D0
0000100C    6000FFF2         BRA.L    $001000
```

Figure 8 Test routine for RAM read/write test

The state listing for this routine is given in Figure 9. A 32-bit port size is indicated by the **DSACK** signals. The routine runs in the supervisor state and repeats itself with the sequence: three sequential program reads (lines +0004 through +0006), one data write (line +0007), one program and data read (lines +0008 and +0009), and another program read (line +0010). This sequence can also be seen in the timing diagram given in Figure 10. The X cursor line corresponds to the line +0000 (SUPERVISOR DATA WRITE) in Figure 10. Function Code signals, **FC2** through **FC0**, either indicate supervisor data space (101) or supervisor program space (110). **DSACK** signals always return a 32-bit port size and only the RAM chip enable signal **RAMCE** is active. Figures 12 and 13 show the timing between the negation of !AS, !DS signals and the negation of **DSACK** signals during a write and read operation, respectively. The specification for this period is 80 ns maximum and it was measured as 70 ns.

68020 - State Listing

Markers: Off

Label > / Base >	ADDR Hex	DATA Hex	STAT Symbol	DSACK Symbol
+0000	00006000	55555555	SUPR DATA WRITE	32 BIT PORT
+0001	0000100C	6000FFF2	SUPR PGRM READ	32 BIT PORT
+0002	00006000	55555555	SUPR DATA READ	32 BIT PORT
+0003	00001010	0C41F0A2	SUPR PGRM READ	32 BIT PORT
+0004	00001000	21FC5555	SUPR PGRM READ	32 BIT PORT
+0005	00001004	55556000	SUPR PGRM READ	32 BIT PORT
+0006	00001008	20386000	SUPR PGRM READ	32 BIT PORT
+0007	00006000	55555555	SUPR DATA WRITE	32 BIT PORT
+0008	0000100C	6000FFF2	SUPR PGRM READ	32 BIT PORT
+0009	00006000	55555555	SUPR DATA READ	32 BIT PORT
+0010	00001010	0C41F0A2	SUPR PGRM READ	32 BIT PORT
+0011	00001000	21FC5555	SUPR PGRM READ	32 BIT PORT
+0012	00001004	55556000	SUPR PGRM READ	32 BIT PORT
+0013	00001008	20386000	SUPR PGRM READ	32 BIT PORT
+0014	00006000	55555555	SUPR DATA WRITE	32 BIT PORT

Figure 9 State listing of the routine for RAM read/write test

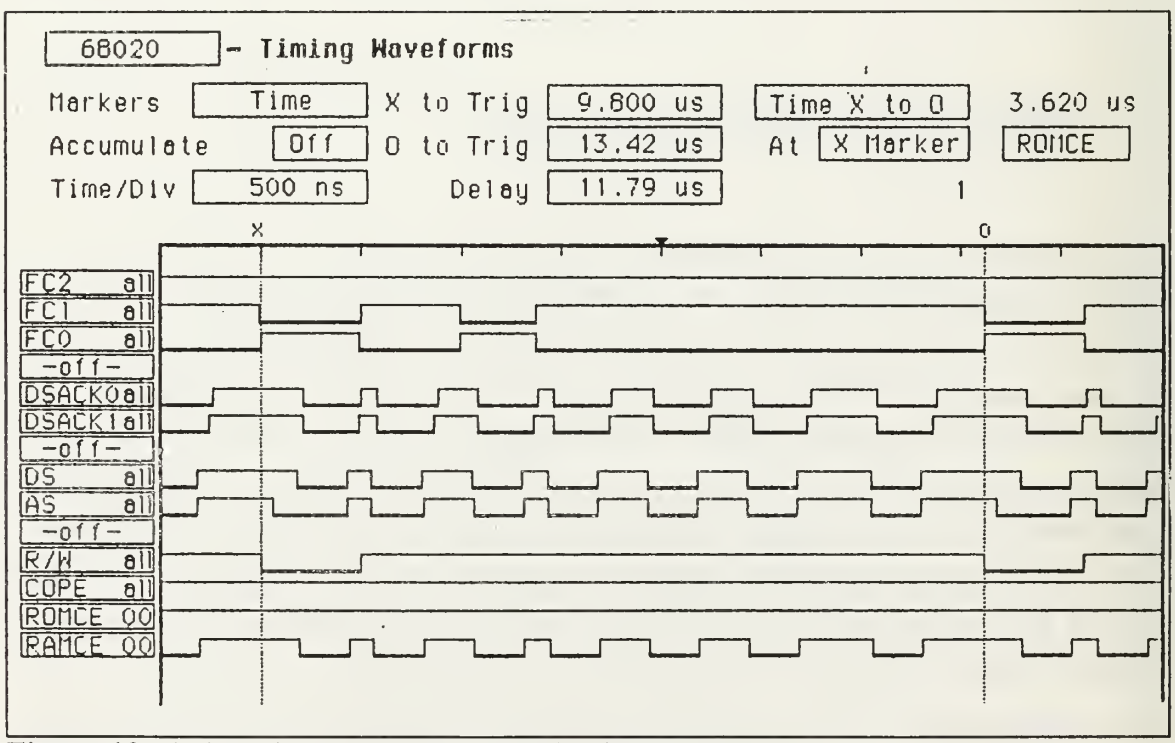


Figure 10 Timing diagram for RAM read/write test.

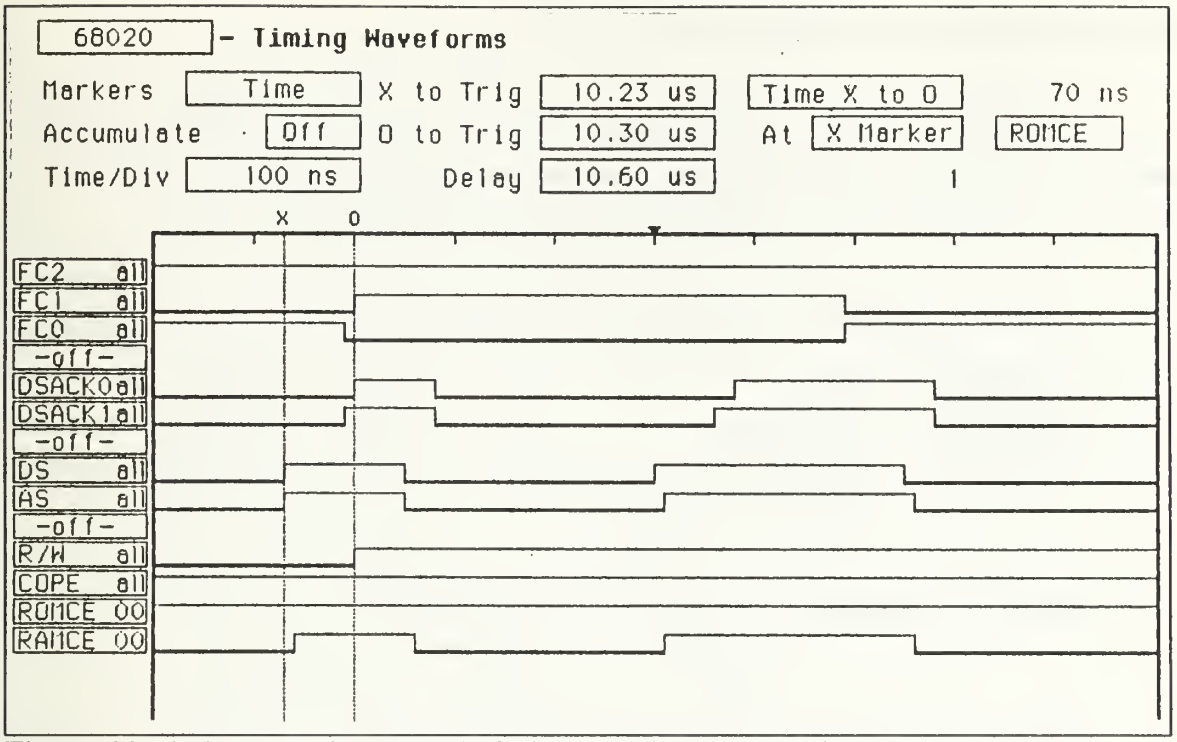


Figure 11 Timing waveforms for !AS, !DS and !DSACK during write operation.

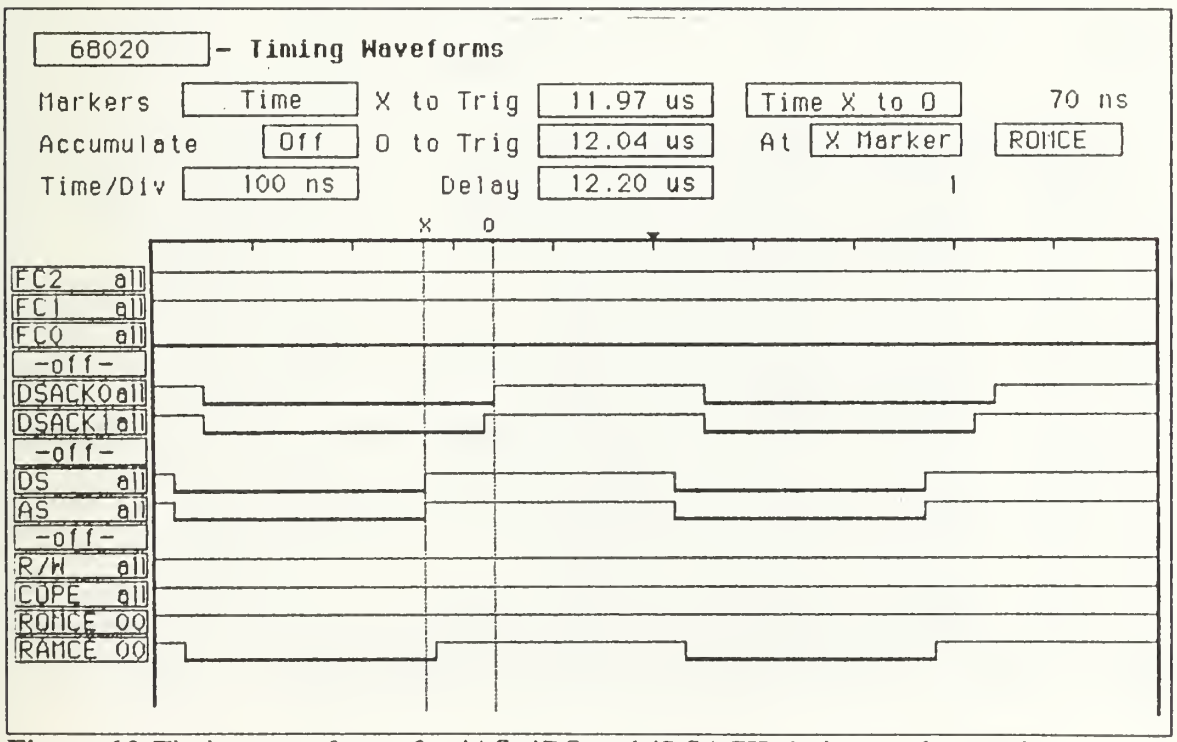


Figure 12 Timing waveforms for !AS, !DS and !DSACK during read operation.

D. Coprocessor communication test

The routine for this test consists of a loop of the instruction FPMOVE #7,FP7 and given in Figure 13. The state listing obtained during the execution of this routine is given in Figure 14, and Figure 15 shows the corresponding timing waveforms.

```
00001000    F23C                    WORD    $F23C
00001002    4380                    CHK.W    D0,D1
00001004    00000007                OR.B     #7,D0
00001008    6000FFF6                BRA.L    $001000
0000100C    6000FFF2                BRA.L    $001000
```

Figure 13 Test routine for coprocessor communication

```
   68020      -   STATE LISTING

Label    >  ADDR      DATA              STAT              DSACK
Base     >   Hex       Hex              Symbol            Symbol

 +0000     00022000 0900FFF6 CPU SPACE                 16 BIT PORT
 +0001     0000100C 6000FFF2 SUPR PGRM READ            32 BIT PORT
 +0002     00001000 F23C4380 SUPR PGRM READ            32 BIT PORT
 +0003     00001004 00000007 SUPR PGRM READ            32 BIT PORT
 +0004     0002200A 43804380 CPU SPACE                 16 BIT PORT
 +0005     00022000 95044380 CPU SPACE                 16 BIT PORT
 +0006     00022010 00000007 CPU SPACE                 32 BIT PORT
 +0007     00001008 6000FFF6 SUPR PGRM READ            32 BIT PORT
 +0008     00022000 0900FFF6 CPU SPACE                 16 BIT PORT
 +0009     0000100C 6000FFF2 SUPR PGRM READ            32 BIT PORT
 +0010     00001000 F23C4380 SUPR PGRM READ            32 BIT PORT
 +0011     00001004 00000007 SUPR PGRM READ            32 BIT PORT
 +0012     0002200A 43804380 CPU SPACE                 16 BIT PORT
 +0013     00022000 95044380 CPU SPACE                 16 BIT PORT
 +0014     00022010 00000007 CPU SPACE                 32 BIT PORT
 +0015     00001008 6000FFF6 SUPR PGRM READ            32 BIT PORT
```

Figure 14 State listing for the routine to test the coprocessor communication.

The execution of the instruction begins by a supervisor program read from the address $1000 (lines +0002 and +0003 in Figure 14). Since this is an F-line instruction, the MC68020 writes to the command CIR, which has an offset $0A (line +0004) and reads the response CIR (line +0005). The response CIR contains the primitive "Evaluate Effective Address and Transfer Data" (code 9504). Then, the MC68020 writes the immediate data into the operand register which has an offset $10 (line +0006). The next read from the response CIR returns a "Null primitive" (code $0900) which shows that the MC68020 is not needed for the execution of the coprocessor instruction, so that the MC68020 can continue to execute the next instruction. The routine loops after executing the branch instruction (line +0007). As it can be seen both in the state listing and the timing waveforms, the port size returned during coprocessor accesses depends on the length of the CIR register being addressed by the MC68881. A 16-bit port size is returned for the response and command CIRs, which are 16-bit wide (lines +0004 and +0005 in Figure 14), and 32-bit port size is returned for the 32-bit wide operand register (line +0006 in Figure 14). As shown in Figure 15, the time between the assertion of coprocessor chip select signal **CopE** and the assertion of **!DS** signal was measured as 50 ns, for which the specification is 35 ns minimum.

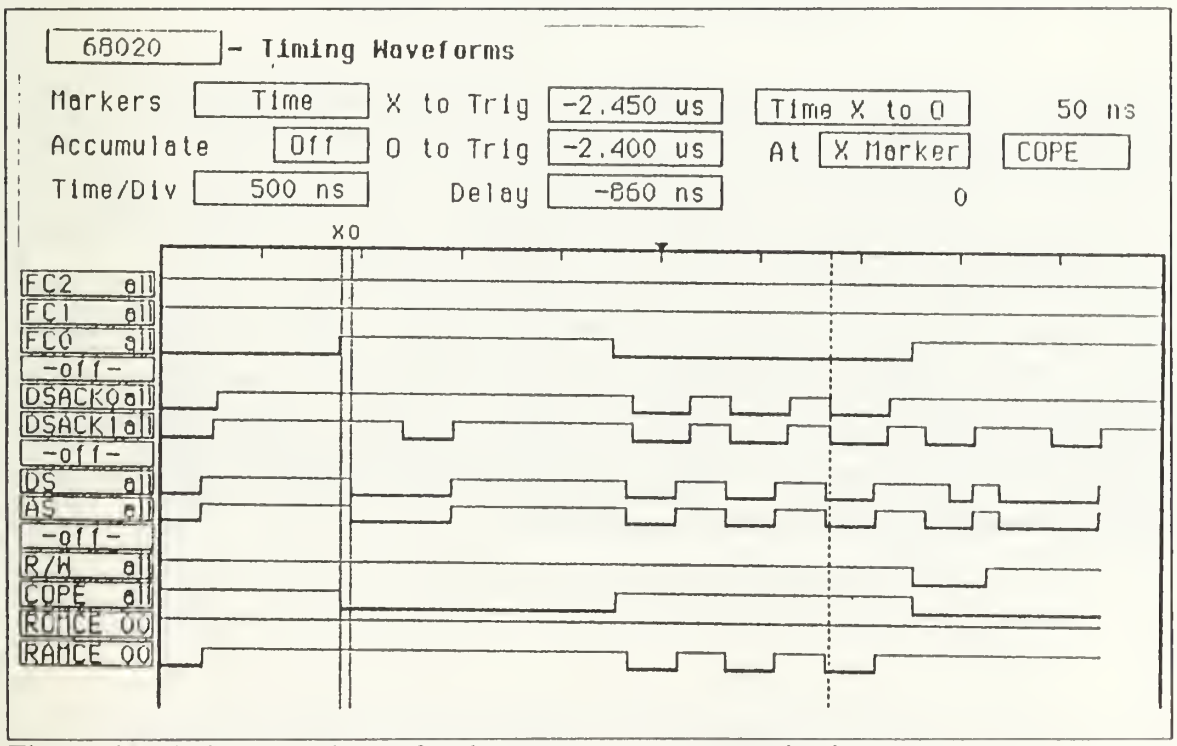


Figure 15 Timing waveforms for the coprocessor communication test

E. Interrupt Level 4 (RS232 communication) test.

The interrupt routine used during this test is the one in Reference 1. The state listing and the timing waveforms are given in Figures 16 and 17, respectively. A level 4 interrupt is generated, when **!IPL2** line is driven low (X marker position in Figure 18). The MC68020 acknowledges the interrupt by setting all the function code lines high (O marker position in Figure 17). During this interrupt acknowledge cycle, the address lines A3 through A1 contain the level of the interrupt being acknowledged, and all the other address lines are driven high (line +0008 in Figure 16). Then, a four-word stack frame (Format $0) is created and the current processor context is saved onto this frame, as follows (refer to the line numbers in Figure 16):

- line +0009 : save the status register.
  (writing a word operand to 32 bit port)
- line +0011 and +0012 : save the program counter.
  (This is also an example of writing a misaligned longword to 32 bit port. Due to misalignment, the MC68020 makes two successive accesses to the stack)
- line +0017 : save the format number and vector offset.
  (writing a word operand to 32 bit port)

The address of the interrupt handler ($00040C08) is fetched from the exception vector address ($00000070) for the level 4 interrupt (line +0010), and the MC68020 enters the interrupt handler routine (line +0013). As it can be seen in the state listing, because of the instruction prefetch, the order of the processor activity does not show the actual order of the instructions executed. For example, the last program read, before the interrupt acknowledge is from the address $000E04E7 (line +0007), but the PC value saved on the stack frame is $000E04E2 (lines +0011 and +0012). This indicates that the MC68020 did not execute the instructions stored in memory locations $000E04E2 and higher, thus the instructions fetched in the lines +0003 through +0007 are only prefetched instructions which were not executed yet.

68020 - State Listing

Markers: Off

Label > / Base >	ADDR Hex	DATA Hex	STAT Symbol	DSACK Symbol
+0003	000E04E3	71000000	SUPR PGRM READ	8 BIT PORT
+0004	000E04E4	4E000000	SUPR PGRM READ	8 BIT PORT
+0005	000E04E5	71000000	SUPR PGRM READ	8 BIT PORT
+0006	000E04E6	4E000000	SUPR PGRM READ	8 BIT PORT
+0007	000E04E7	F9000000	SUPR PGRM READ	8 BIT PORT
+0008	FFFFFFF9	00000000	CPU SPACE	WAIT STATE
+0009	0001BD54	20002000	SUPR DATA WRITE	32 BIT PORT
+0010	00000070	00040C08	SUPR DATA READ	32 BIT PORT
+0011	0001BD56	000E000E	SUPR DATA WRITE	32 BIT PORT
+0012	0001BD58	04E204E2	SUPR DATA WRITE	32 BIT PORT
+0013	00040C08	02E204E2	SUPR PGRM READ	8 BIT PORT
+0014	00040C09	AFE204E2	SUPR PGRM READ	8 BIT PORT
+0015	00040C0A	FFE204A2	SUPR PGRM READ	8 BIT PORT
+0016	00040C0B	F5000402	SUPR PGRM READ	8 BIT PORT
+0017	0001BD5A	00700070	SUPR DATA WRITE	32 BIT PORT

Figure 16 State listing for the interrupt level 4 test

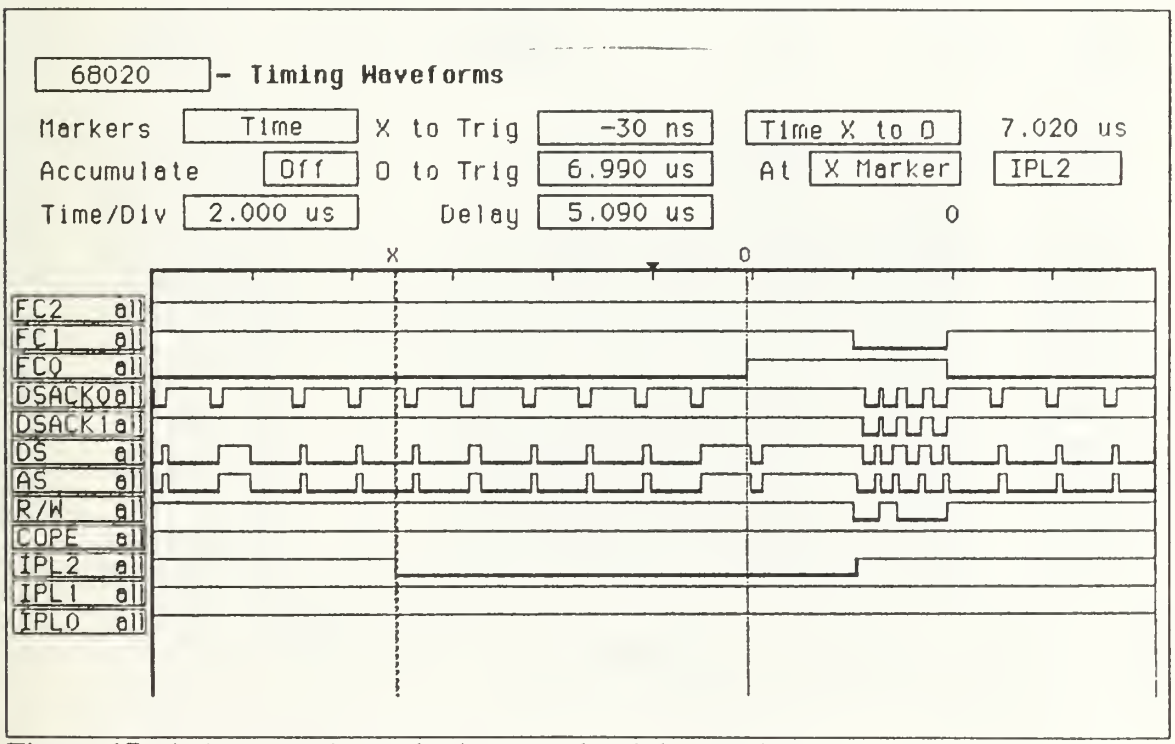


Figure 17 Timing waveforms for interrupt level 4 operation.

F. Interrupt Level 6 (Software Abort) test.

The interrupt routine used in this test is the one in Reference 1. The state listing and timing waveforms are given in Figures 18 and 19, respectively. The interrupt is handled in the same way as the level 4 interrupt. The only difference is the level of the interrupt and the exception vector address ($00000078). The interrupt is generated by driving both **!IPL2** and **!IPL1** lines low, simultaneously (See Figure 19). The MC68020 does not ackowledge the interrupt, immediately. Instead, it drives the **!IPEND** line low and completes the current instruction execution. When the current instruction is completed, the MC68020 enters the interrupt acknowledge cycle and negates the **!IPEND** line (X marker position in Figure 19). Address line A18 is shown as a sample of the address bus, during this activity. It is asserted first during the interrupt acknowledge cycle and second to access the routine in the ROM.

68020 - STATE LISTING

Label Base	> ADDR Hex	DATA Hex	STAT Symbol	DSACK Symbol
+0000	00022000	95044380	CPU SPACE	16 BIT PORT
+0001	00022010	00000007	CPU SPACE	32 BIT PORT
+0002	00001008	6000FFF6	SUPR PGRM READ	32 BIT PORT
+0003	00022000	0900FFF6	CPU SPACE	16 BIT PORT
+0004	0000100C	6000FFF2	SUPR PGRM READ	32 BIT PORT
+0005	FFFFFFFD	0000FFF2	CPU SPACE	WAIT STATE
+0006	0001FFD4	20042004	SUPR DATA WRITE	32 BIT PORT
+0007	00000078	00040CCC	SUPR DATA READ	32 BIT PORT
+0008	0001FFD6	00000000	SUPR DATA WRITE	32 BIT PORT
+0009	0001FFD8	10081008	SUPR DATA WRITE	32 BIT PORT
+0010	00040CCC	00081008	SUPR PGRM READ	8 BIT PORT
+0011	00040CCD	7C081008	SUPR PGRM READ	8 BIT PORT
+0012	00040CCE	07081008	SUPR PGRM READ	8 BIT PORT
+0013	00040CCF	00081008	SUPR PGRM READ	8 BIT PORT
+0014	0001FFDA	00780078	SUPR DATA WRITE	32 BIT PORT

Figure 18 The state listing for interrupt level 6 test.

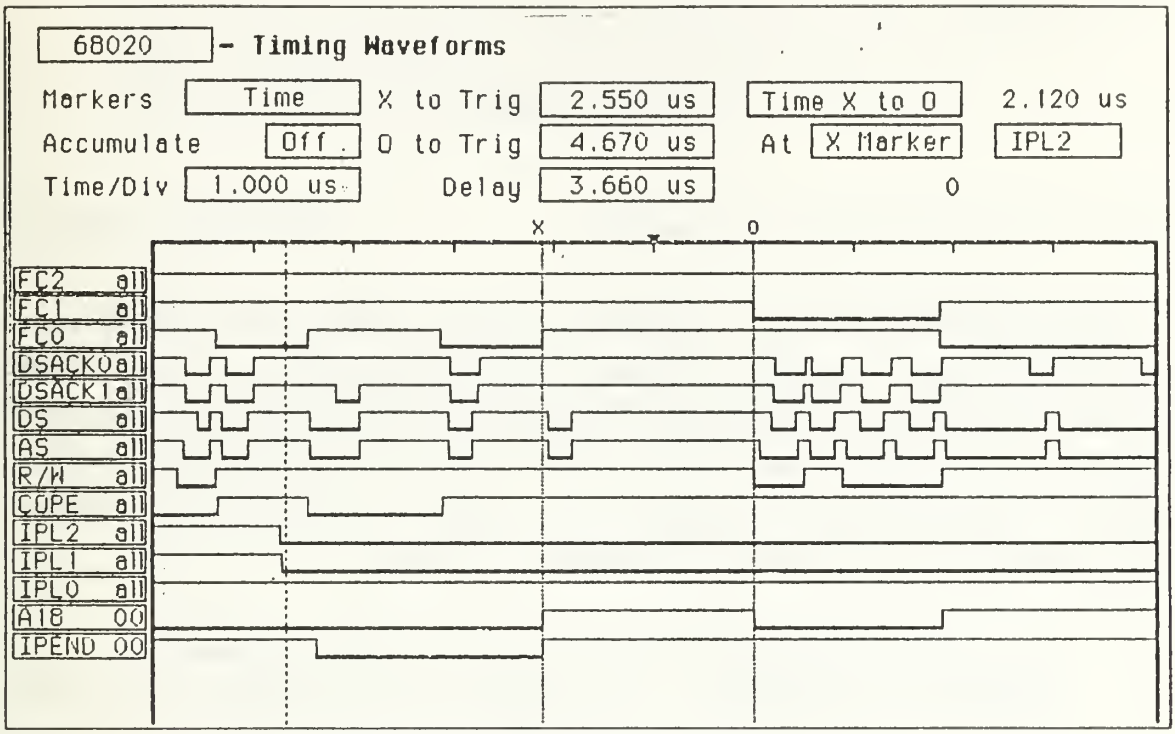


Figure 19 Timing waveforms for interrupt level 6 test.

V. CONCLUSIONS

A. The current implementation of the ECB

The ECB designed and implemented in this thesis can be used as a state-of-the-art tool for teaching and research. The ECB requires an easy-to-install firmware for initializing and handling the communication with a control device (see Reference 1). The result of this effort is a new and powerful microprocessor which is simple. In comparison with the ECB previously designed by Motorola (using M68000 microprocessor), the new design has the following advantages;

- The number of components on the board is less. (10 components - not including the components for the external I/O device interface. The Motorola ECB has 61 components).

- Availability of the coprocessor which provides a very fast computation mechnism for floating point operations. It can also be used as a tool for teaching.

- Higher clock speed (quadrupled to 16 MHz).

- Only one intelligent terminal is required to run the board. (the Motorola ECB requires two, one intelligent terminal to assemble and download user programs and one dumb terminal to run the downloaded program).

It has the following disadvantages.

- Slow rate of response to user commands. (This is due to the fact that the user commands and the result of these commands have to be passed back and forth between the board and the intelligent terminal via the 9600 baud rate RS-232 interface. This is a trade-off between the number of components/terminals and the speed. Transmission or reception of a byte takes 10.4 ms with a 9600 baud rate).

- Cache memory has not been enabled. (Disabling the cache memory allowed us to monitor the external bus activity during development of the ECB and helped troubleshoot the failures and learn the details of processor operation during implementation).

B. Future Improvements

The ECB has a provision to install TTL series 74244 and 74245 line drivers/ receivers for an external I/O device (8 bit). All the pads and holes are available to install the line drivers/receivers. The connection diagram is given in **Appendix G.** The byte I/O feature has not been implemented and tested.

APPENDIX A: MC68020 SIGNAL DESCRIPTION

This section describes the function of each individual signal or group of signals and their utilization on the ECB.

1. Function Code Signals (FC0 through FC2)

- Three-state outputs.
- Identify the processor and address space of the current bus cycle, as shown in Table 2.

Table 2 Function Code Encodings

FC2	FC1	FC0	ADDRESS SPACE
0	0	0	UNDEFINED
0	0	1	USER DATA SPACE
0	1	0	USER PROGRAM SPACE
0	1	1	UNDEFINED
1	0	0	UNDEFINED
1	0	1	SUPERVISOR DATA SPACE
1	1	0	SUPERVISOR PROGRAM SPACE
1	1	1	CPU SPACE

- Not utilized on the ECB.

2. Address Bus Signals (A0 through A31)

- Three state outputs.
- Provide the address, up to 4 gigabytes, for a bus transfer in all address spaces except for CPU space in which the bus specifies CPU related information.
- On the ECB;

  A18, A17, A15 generate chip enable signals for coprocessor and memory.

  A19, A15 are used to transmit RS-232 data.

A19, A17 are used to receive RS-232 data.

A1, A0 are used to generate read/write signals for RAM.

A14 through A0 are used to address ROM.

A14 through A2 are used to address RAM.

A31 through A20 are not used.

3. Data Bus Signals (D0 through D31)

- Three state inputs/outputs.
- Provides exchange of data between MC68020 and external devices.
- On the ECB;

  D31 through D24 are connected to ROM.

  D31 through D0 are connected to RAM and coprocessor.

4. Transfer Size Signals (SIZ0, SIZ1)

- Three state outputs.
- Indicate the remaining number of bytes of an operand to be transfered in a bus cycle, as shown in Table 3.

Table 3 Transfer Size Encodings

TRANSFER SIZE	SIZ1	SIZ0
BYTE	0	1
WORD	1	0
3 BYTE	1	1
LONG WORD	0	0

- On the ECB;

  SIZ0, SIZ1 are used to generate read/write signals for RAM.

5. External Cycle Start (!ECS)

- Output
- In case of a cache miss, indicates the start of an external bus cycle, if validated by Address Strobe (!AS) later.
- Not utilized on the ECB.

6. Operand Cycle Start (!OCS)

- Output
- Indicates the start of an instruction prefetch or an operand transfer with the same restrictions as in !ECS.
- Not utilized on the ECB.

7. Read-Modify-Write Cycle (!RMC)

- Three state output
- Indicates an indivisible read modify write cycle on the bus.
- Not utilized on the ECB.

8. Address Strobe (!AS)

- Three state output
- Indicates the availability of valid function code, address, size, and read/write information on the bus.
- On the ECB;
  Used as a synchronization pulse in the generation of DSACK0, DSACK1, and PHANTOM signals.

9. Data Strobe (!DS)

- Three state output
- In a write cycle, indicates that valid data is available on the data bus.
  In a read cycle, signals the slave device to drive the data bus.
- On the ECB;
  Used to generate chip select and read/write signals for memory.

10. Read/Write (R/!W)

- Three state output
- High level on this output indicates a read from an external device,
  Low level indicates a write to an external device.
- On the ECB;
  Used to generate chip select, read/write and output enable signals for memory.

11. Data Buffer Enable (!DBEN)

- Three state output
- Provides an enable to external data buffers.
- Not utilized on the ECB.

12. Data Transfer and Size Acknowledge (!DSACK0, !DSACK1)

- Inputs
- Indicates the port size of the external device and the completion of the data transfer, as shown in Table 4.

Table 4 DSACK codes.

$\overline{DSACK1}$	$\overline{DSACK0}$	BUS CYCLE STATUS
1	1	INSERT WAIT STATES
1	0	8 BIT PORT - CYCLE COMPLETED
0	1	16 BIT PORT - CYCLE COMPLETED
0	0	32 BIT PORT - CYCLE COMPLETED

- On the ECB;
  Indicate an 8-bit port size for ROM, 32-bit port size for RAM, and coprocessor.

13. Cache Disable (!CDIS)

- Input
- Allows to enable/disable the on-chip cache memory.
- On the ECB;
  Pulled down to ground to disable the cache.

14. Interrupt Priority Level Signals (!IPL0, !IPL1, !IPL2)

- Inputs
- Indicate the level of the interrupt requested by an external device, as shown in Table 5.

Table 5 Interrupt Priority and mask levels

			LEVEL	
$\overline{IPL2}$	$\overline{IPL1}$	$\overline{IPL0}$	REQUESTED	MASK
1	1	1	0	N/A
1	1	0	1	0
1	0	1	2	0-1
1	0	0	3	0-2
0	1	1	4	0-3
0	1	0	5	0-4
0	0	1	6	0-5
0	0	0	7	0-7

- On the ECB;
  Interrupt level 4 (!IPL2) is used for RS-232 communication.
  Interrupt level 6 (!IPL2, !IPL1) is used for software abort.

15. Interrupt Pending (!IPEND)

- Output
- Indicates that the active interrupt priority level is higher than the level of the interrupt mask in the status register or indicates the recognition of a non-maskable interrupt.
- Not utilized on the ECB.

16. Autovector (!AVEC)

- Input
- When asserted, interrupt vector is generated internally during an interrupt acknowledge cycle.
- On the ECB;
  All interrupts are autovectored.

17. Bus Request (!BR)

- Input
- Indicates that some device other than MC68020 has a request to become a bus master.
- Not utilized on the ECB.

18. Bus Grant (!BG)

- Output
- Indicates that MC68020 will release the bus upon the completion of the current bus cycle for use by the device issuing a Bus Request.
- Not utilized on the ECB.

19. Bus Grant Acknowledge (!BGACK)

- Input
- Indicates that some device other than MC68020 has become a bus master.
- Not utilized on the ECB.

20. Reset (!RESET)

- Open drain input and output.
- When used as an input, MC68020 enters reset exception processing;
  when used as an output, external devices are reset, and no internal action is taken.
- On the ECB;
  Used as an input only to reset the processor during power up or reset by the user.

21. Halt (!HALT)

- Open drain input and output.
- When used as an input, MC68020 halts;
  previous bus cycle information is kept on read/write, function code, size signals and on the address bus.
  The data bus stays in high impedance state.
  All control signals stay inactive.
  When used as an output, signals the external devices that MC68020 has halted.

- On the ECB;
  Asserted at the same time as the reset input, during power up or reset.

22. Bus Error (!BERR)

- Input
- Indicates a problem with the current bus cycle.
- Not utilized on the ECB.

23.Clock (CLK)

- TTL-compatible input
- On the ECB;
  16 MHz clock is applied to this input.

Table 6 MC68020 AC Electrical Characteristics. (Copied from Reference 2)

Num	Characteristic	Symbol	MC68020RC12 Min	MC68020RC12 Max	MC68020RC16 Min	MC68020RC16 Max	Unit
6	Clock High to Address/FC/Size/$\overline{RMC}$ Valid	t_{CHAV}	0	40	0	30	ns
6A	Clock High to $\overline{ECS}$, $\overline{OCS}$ Asserted	t_{CHEV}	0	30	0	20	ns
7	Clock High to Address, Data, FC, $\overline{RMC}$, Size High Impedance	t_{CHAZx}	0	80	0	60	ns
8	Clock High to Address/FC/Size/$\overline{RMC}$ Invalid	t_{CHAZn}	0	–	0	–	ns
9	Clock Low to $\overline{AS}$, $\overline{DS}$ Asserted	t_{CLSA}	3	40	3	30	ns
9A[1]	$\overline{AS}$ to $\overline{DS}$ Assertion (Read) (Skew)	t_{STSA}	–20	20	–15	15	ns
10	$\overline{ECS}$ Width Asserted	t_{ECSA}	20	–	20	–	ns
10A	$\overline{OCS}$ Width Asserted	t_{OCSA}	25	–	20	–	ns
11[6]	Address/FC/Size/$\overline{RMC}$ Valid to $\overline{AS}$ (and $\overline{DS}$ Asserted Read)	t_{AVSA}	20	–	15	–	ns
12	Clock Low to $\overline{AS}$, $\overline{DS}$ Negated	t_{CLSN}	0	40	0	30	ns
12A	Clock Low to $\overline{ECS}$/$\overline{OCS}$ Negated	t_{CLEN}	0	40	0	30	ns
13	$\overline{AS}$, $\overline{DS}$ Negated to Address, FC, Size Invalid	t_{SNAI}	20	–	15	–	ns
14	$\overline{AS}$ (and $\overline{DS}$ Read) Width Asserted	t_{SWA}	120	–	100	–	ns
14A	$\overline{DS}$ Width Asserted Write	t_{SWAW}	50	–	40	–	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	t_{SN}	50	–	40	–	ns
18	Clock High to $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, $\overline{DBEN}$ High Impedance	t_{CSZ}	–	80	–	60	ns
17[6]	$\overline{AS}$, $\overline{DS}$ Negated to R/$\overline{W}$ High	t_{SNRN}	20	–	15	–	ns
18	Clock High to R/$\overline{W}$ High	t_{CHRH}	0	40	0	30	ns
20	Clock High to R/$\overline{W}$ Low	t_{CHRL}	0	40	0	30	ns
21[6]	R/$\overline{W}$ High to $\overline{AS}$ Asserted	t_{RAAA}	20	–	15	–	ns
22[6]	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	t_{RASA}	90	–	20	–	ns
23	Clock High to Data Out Valid	t_{CHDO}	–	40	–	30	ns
25[6]	$\overline{DS}$ Negated to Data Out Invalid	t_{SNDI}	20	–	15	–	ns
26[6]	Data Out Valid to $\overline{DS}$ Asserted (Write)	t_{DVSA}	20	–	15	–	ns
27	Data-In Valid to Clock Low (Data Setup)	t_{DICL}	10	–	5	–	ns
27A	Late $\overline{BERR}$/$\overline{HALT}$ Asserted to Clock Low Setup Time	t_{BELCL}	25	–	20	–	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$, $\overline{BERR}$, $\overline{HALT}$, $\overline{AVEC}$ Negated	t_{SNDN}	0	110	0	80	ns
29	$\overline{DS}$ Negated to Data-In Invalid (Data-In Hold Time)	t_{SNDI}	0	–	0	–	ns
29A	$\overline{DS}$ Negated to Data-In (High Impedance)	t_{SNDI}	–	80	–	60	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid	t_{DADI}	–	60	–	60	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid ($\overline{DSACK}$ Asserted Skew)	t_{DADV}	–	20	–	15	ns
32	$\overline{RESET}$ Input Transition Time	t_{HRrf}	–	2.5	–	2.5	Clk Per
33	Clock Low to $\overline{BG}$ Asserted	t_{CLBA}	0	40	0	30	ns
34	Clock Low to $\overline{BG}$ Negated	t_{CLBN}	0	40	0	30	ns
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	t_{BRAGA}	1.5	3.5	1.5	3.5	Clk Per
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	t_{GAGN}	1.5	3.5	1.5	3.5	Clk Per
39	$\overline{BG}$ Width Negated	t_{GN}	120	–	90	–	ns
39A	$\overline{BG}$ Width Asserted	t_{GA}	120	–	90	–	ns
40	Clock High to $\overline{DBEN}$ Asserted (Read)	t_{CHDAR}	0	40	0	30	ns
41	Clock Low to $\overline{DBEN}$ Negated (Read)	t_{CLDNR}	0	40	0	30	ns
42	Clock Low to $\overline{DBEN}$ Asserted (Write)	t_{CLDAW}	0	40	0	30	ns
43	Clock High to $\overline{DBEN}$ Negated (Write)	t_{CHDNW}	0	40	0	30	ns
44[6]	R/$\overline{W}$ Low to $\overline{DBEN}$ Asserted (Write)	t_{RADA}	20	–	15	–	ns
45[5]	$\overline{DBEN}$ Width Asserted Read Write	t_{DA}	80 160	– –	60 120	– –	ns ns
46	R/$\overline{W}$ Width Asserted (Write or Read)	t_{RWA}	180	–	150	–	ns
47a	Asynchronous Input Setup Time	t_{AIST}	10	–	5	–	ns
47b	Asynchronous Input Hold Time	t_{AIHT}	20	–	15	–	ns
48[4]	$\overline{DSACKx}$ Asserted to $\overline{BERR}$, $\overline{HALT}$ Asserted	t_{DABA}	–	35	–	30	ns
53	Data Out Hold from Clock High	t_{DOCH}	0	–	0	–	ns
55	R/$\overline{W}$ Asserted to Data Bus Impedance Change	t_{RADC}	40	–	40	–	ns
56	$\overline{RESET}$ Pulse Width (Reset Instruction)	t_{HRPW}	512	–	512	–	Clks
57	$\overline{BERR}$ Negated to $\overline{HALT}$ Negated (Rerun)	t_{BNHN}	0	–	0	–	ns

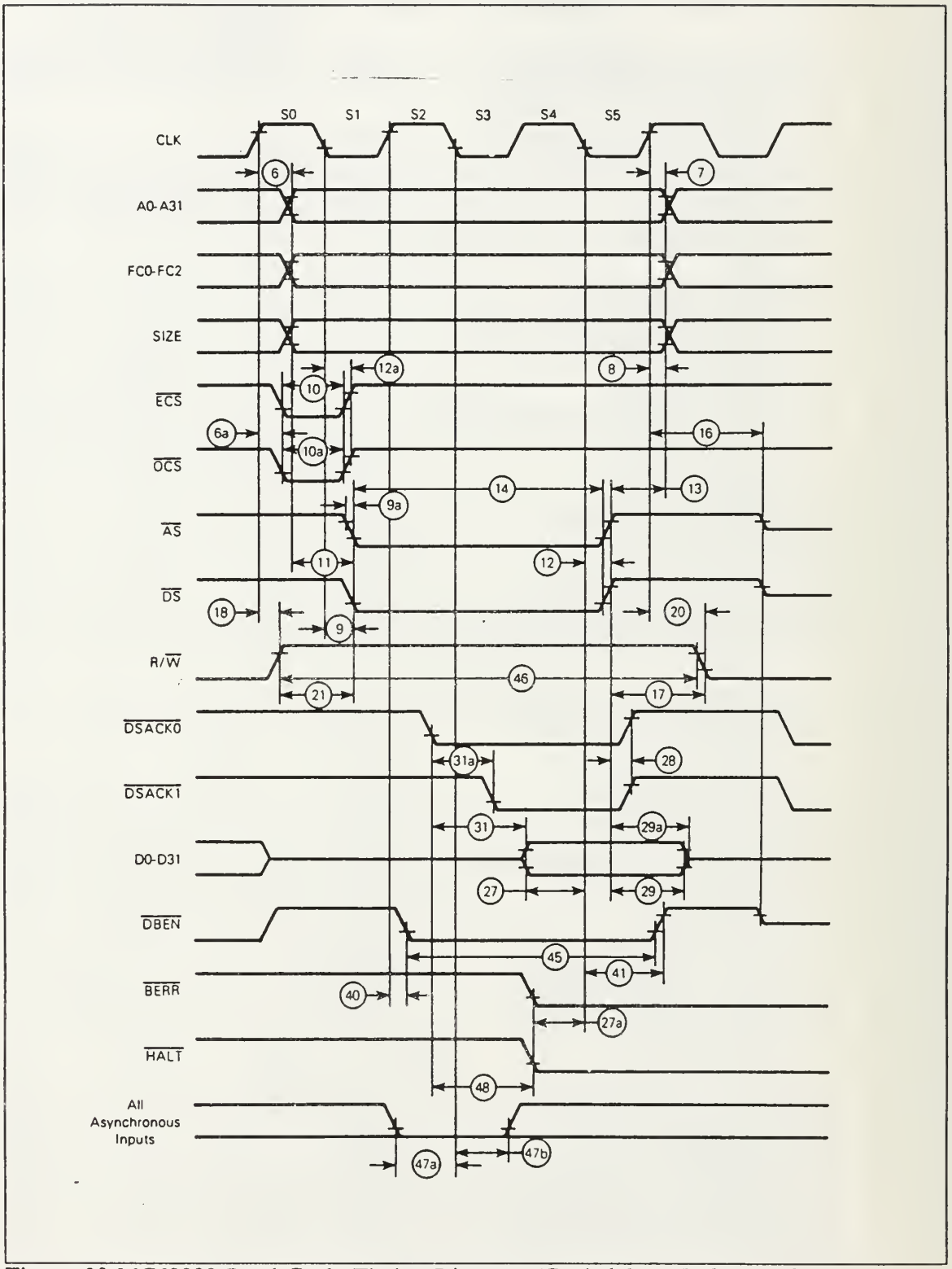


Figure 20 MC68020 Read Cycle Timing Diagram. (Copied from Reference 2)

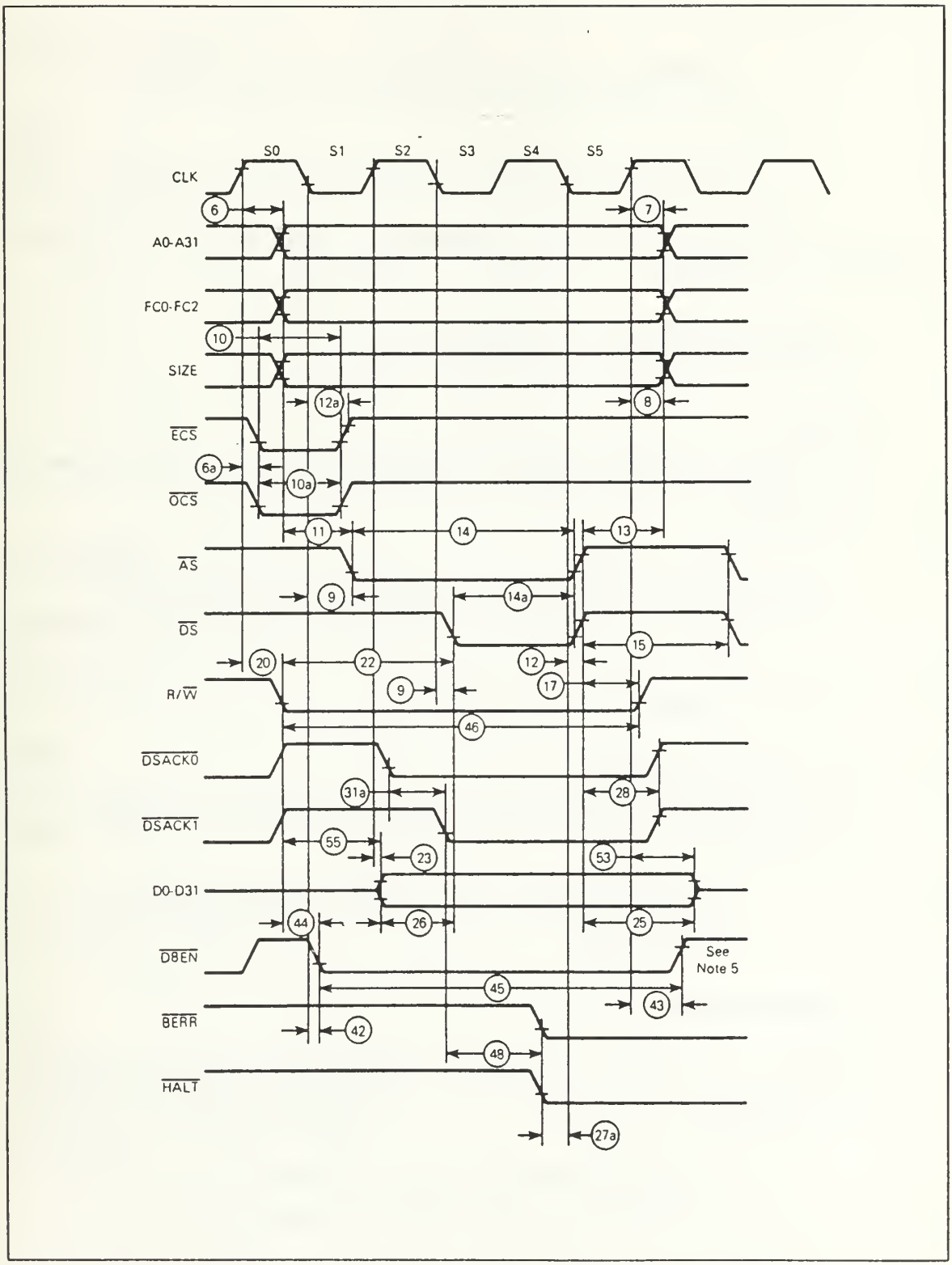


Figure 21 MC68020 Write Cycle Timing Diagram. (Copied from Reference 2)

APPENDIX B: MC68881 SIGNAL DESCRIPTION

This section describes the function of each individual signal and its utilization on the ECB.

1. Address Bus Signals (A0 through A4)

- Inputs.
- Are used by the main processor to access any coprocessor interface register in the CPU address space. A0 is used to configure the data bus size.
- On the ECB;

  A0 is connected to high in order to configure a 32 bit bus connection.

  A1 through A4 are connected to corresponding address bus pins of MC68020.

2. Data Bus Signals (D0 through D31)

- Three state inputs/outputs.
- Provides exchange of data between MC68881 and the main processor.
- On the ECB;

  D31 through D0 are connected to corresponding data bus pins of the MC68020.

3. Address Strobe (!AS)

- Input
- Indicates the availability of valid address, chip select and read/write signals.
- On the ECB;

  is directly connected to !AS pin of MC68020.

4. Size Signal (!SIZE)

- Input.
- Used in conjunction with A0 configure the data bus size as follows:

Table 7 MC68881 Data Bus Size Encoding.

A0	SIZE	DATA BUS SIZE
X	0	8 BIT
0	1	16 BIT
1	1	32 BIT

- On the ECB;

  is connected to high in order to configure a 32 bit bus connection.

5. Chip Select (!CS)

- Input
- Enables the main processor access to the coprocessor interface registers.
- On the ECB;

  is generated by the address bits A18, A17 and A15.

6. Read/Write (R/!W)

- Input
- Indicates the direction of bus activity.
  Low level: a read from MC68881.
  High level: a write to MC68881.
- On the ECB;

  is directly connected to R/!W output of MC68020.

7. Data Strobe (!DS)

- Input
- Indicates a valid data on the data bus, during a write cycle.
- On the ECB;
  is directly connected to !DS pin of MC68020.

8. Data Size And Acknowledge (!DSACK0, !DSACK1)

- Three state output
- Indicates the port size of the coprocessor interface and the completion of the bus cycle to the main processor.
- On the ECB;

  are directly connected to !DSACK0 and !DSACK1 pins of MC68020. They report a port size of 32 bits to the main processor.

9. Reset (!RESET)

- Input.
- Initializes the floating point data registers and clears the floating point control, status and instruction address registers.
- On the ECB;

  is connected to the same reset circuit as MC68020.

10. Sense Device (!SENSE)

- Output.
- Can be utilized as an indication to the presence of MC68881.
- Not utilized on the ECB

11. Clock (CLK)

- TTL compatible input
- On the ECB;
  MC68020 clock is applied to this input.

Table 8 MC68881 AC Electrical Characteristics. (Copied from Reference 3)

No.	Characteristic	Symbol	MC68881RC12		MC68881RC16		Unit
			Min	Max	Min	Max	
6	Address Valid to $\overline{AS}$ Asserted (Note 5)	t_{AVASL}	20	—	15	—	ns
6a	Address Valid to $\overline{DS}$ Asserted (Read) (Note 5)	t_{AVRDSL}	20	—	15	—	ns
6b	Address Valid to $\overline{DS}$ Asserted (Write) (Note 5)	t_{AVWDSL}	65	—	50	—	ns
7	$\overline{AS}$ Negated to Address Invalid (Note 6)	t_{ASHAX}	15	—	10	—	ns
7a	$\overline{DS}$ Negated to Address Invalid (Note 6)	t_{DSHAX}	15	—	10	—	ns
8	$\overline{CS}$ Asserted to $\overline{AS}$ Asserted or $\overline{AS}$ Asserted to $\overline{CS}$ Asserted (Note 8)	t_{CVASL}	0	—	0	—	ns
8a	$\overline{CS}$ Asserted to $\overline{DS}$ Asserted or $\overline{DS}$ Asserted to $\overline{CS}$ Asserted (Read) (Note 9)	t_{CVRDSL}	0	—	0	—	ns
8b	$\overline{CS}$ Asserted to $\overline{DS}$ Asserted (Write)	t_{CVWDSL}	45	—	35	—	ns
9	$\overline{AS}$ Negated to $\overline{CS}$ Negated	t_{ASHCX}	10	—	10	—	ns
9a	$\overline{DS}$ Negated to $\overline{CS}$ Negated	t_{DSHCX}	10	—	10	—	ns
10	R/$\overline{W}$ High to $\overline{AS}$ Asserted (Read)	t_{RVASL}	20	—	15	—	ns
10a	R/$\overline{W}$ High to $\overline{DS}$ Asserted (Read)	t_{RVDSL}	20	—	15	—	ns
10b	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	t_{RLSL}	45	—	35	—	ns
11	$\overline{AS}$ Negated to R/$\overline{W}$ Low (Read) or $\overline{AS}$ Negated to R/$\overline{W}$ High (Write)	t_{ASHRX}	15	—	10	—	ns
11a	$\overline{DS}$ Negated to R/$\overline{W}$ Low (Read) or $\overline{DS}$ Negated to R/$\overline{W}$ High (Write)	t_{DSHRX}	15	—	10	—	ns
12	$\overline{DS}$ Width Asserted (Write)	t_{DSL}	50	—	40	—	ns
13	$\overline{DS}$ Width Negated	t_{DSH}	50	—	40	—	ns
13a	$\overline{DS}$ Negated to $\overline{AS}$ Asserted (Note 4)	t_{DSHASL}	40	—	30	—	ns
14	$\overline{CS}$, $\overline{DS}$ Asserted to Data-Out Valid (Read) (Note 2)	t_{DSLDO}	—	110	—	80	ns
15	$\overline{DS}$ Negated to Data-Out Invalid (Read)	t_{DSHDO}	0	—	0	—	ns
16	$\overline{DS}$ Negated to Data-Out High Impedance (Read)	t_{DSHDZ}	—	70	—	50	ns
17	Data-In Valid to $\overline{DS}$ Asserted (Write)	t_{DIDSL}	20	—	15	—	ns
18	$\overline{DS}$ Negated to Data-In Invalid (Write)	t_{DSHDI}	20	—	15	—	ns
19	$\overline{START}$ True to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted (Notes 2,10)	t_{SLDAL}	—	70	—	50	ns
19a	$\overline{DSACK0}$ Asserted to $\overline{DSACK1}$ Asserted (Skew) (Note 7)	t_{DADAS}	−20	20	−15	15	ns
20	$\overline{DSACK0}$ or $\overline{DSACK1}$ Asserted to Data-Out Valid (Read)	t_{DALDO}	—	60	—	50	ns
21	$\overline{START}$ False to $\overline{DSACK0}$ and $\overline{DSACK1}$ Negated (Note 10)	t_{SHDAH}	—	70	—	50	ns
22	$\overline{START}$ False to $\overline{DSACK0}$ and $\overline{DSACK1}$ High Impedance (Note 10)	t_{SJDAZ}	—	90	—	70	ns
23	$\overline{START}$ True to Clock High (Synchronous Read) (Notes 3, 10)	t_{DSLCH}	0	—	0	—	ns
24	Clock Low to Data-Out Valid (Synchronous Read) (Note 3)	t_{CLDO}	—	140	—	105	ns
25	$\overline{START}$ True to Data-Out Valid (Synchronous Read) (Notes 3, 10, and 11)	t_{DSSLDO}	1.5 Clks	140 + 2.5 Clks	1.5 Clks	105 + 2.5 Clks	ns
26	Clock Low to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted (Synchronous Read) (Note 3)	t_{CLDAL}	—	100	—	75	ns
27	$\overline{START}$ True to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted (Synchronous Read) (Notes 3, 10, and 11)	t_{DSLDAL}	1.5 Clks	100 + 2.5 Clks	1.5 Clks	75 + 2.5 Clks	ns

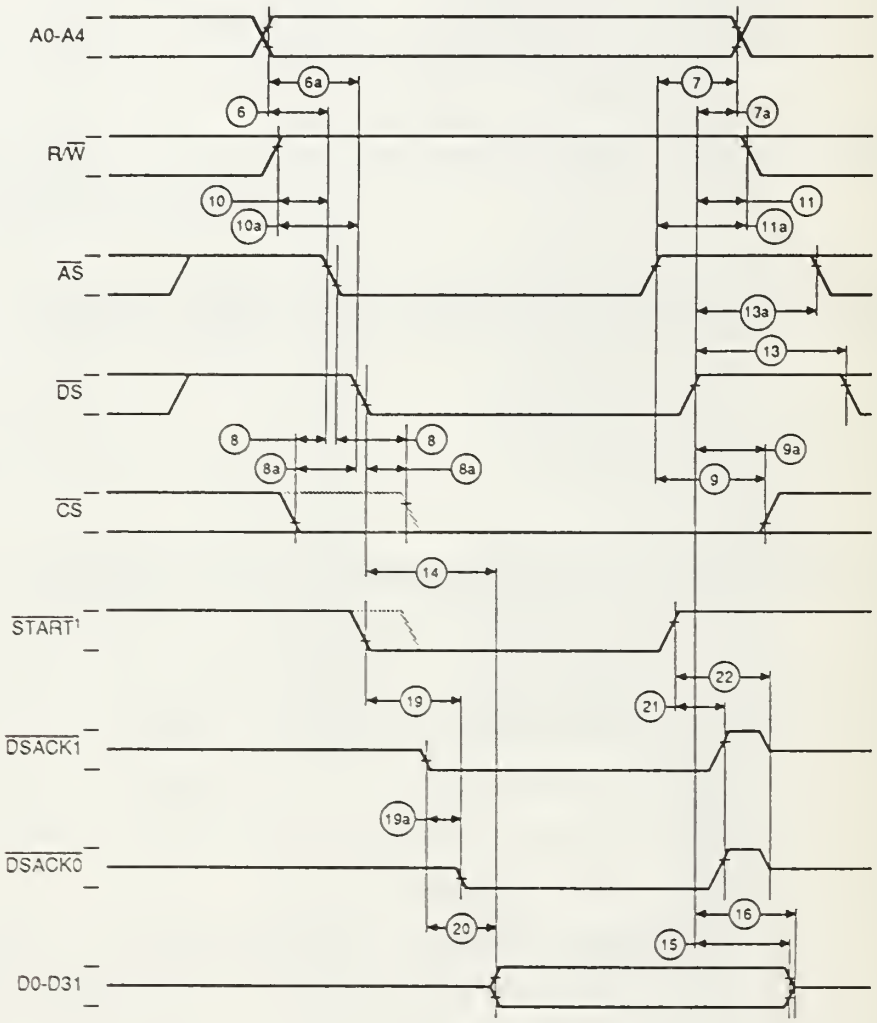


Figure 22 MC68881 Read Cycle Timing Diagram. (Copied from Reference 3)

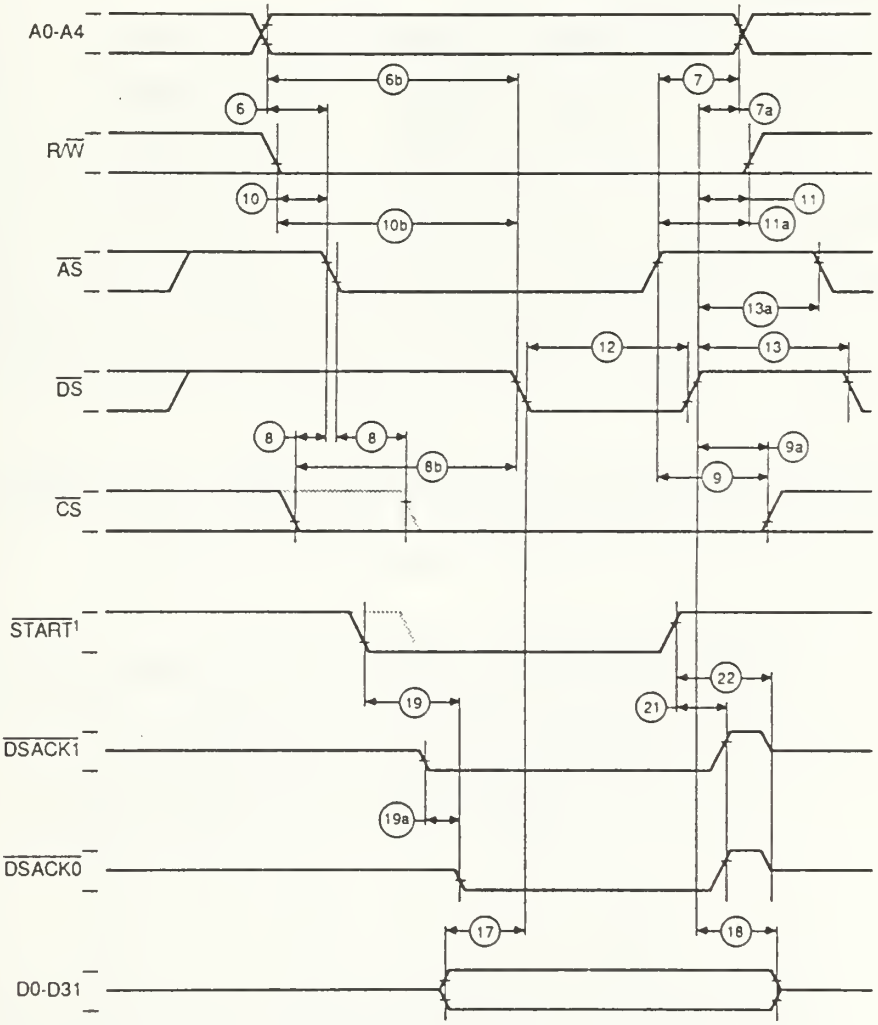


Figure 23 MC68881 Write Cycle Timing Diagram (Copied from Reference 3)

APPENDIX C: MC68020 BUS OPERATION

This section describes the basic bus operation of MC68020.

1. Operand Transfers

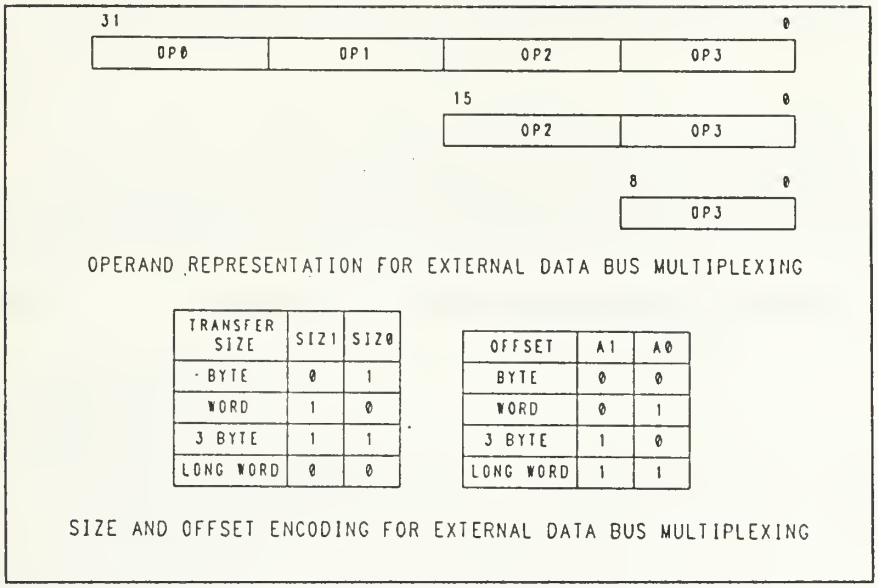
Unlike the old M68000 family members, there is no restriction on the alignment of data in memory, but the instruction alignment on word boundaries is enforced in order to obtain maximum efficiency. MC68020 can transfer byte, word, and longword operands to/from 8, 16, and 32-bit data ports signalled by the data transfer and size acknowledge (!DSACK0, !DSACK1) inputs. A 32-bit port uses all data lines D31 through D0. Communication with a 16-bit port is provided over D31 through D16, and with an 8-bit port over D31 through D24. It takes MC68020 one bus cycle to fetch a long word from a 32-bit port, two bus cycles from a 16-bit port and four bus cycles from an 8-bit port. The bytes of an operand of any size can be routed to any byte position of 32-bit data bus, according to the size outputs and the address lines A0 and A1. By the use of this scheme, the operand alignment restriction is eliminated. Table 9 shows how the bytes of an operand is multiplexed on the data bus.

Table 9 MC68020 External Data Bus Multiplexing.

TRANSFER SIZE	SIZE		ADDRESS		OPERAND POSITION			
	SIZ1	SIZ0	A1	A0	D31:D24	D23:D16	D15:D8	D7-D0
BYTE	0	1	X	X	OP3	OP3	OP3	OP3
WORD	1	0	X	0	OP2	OP3	OP2	OP3
	1	0	X	1	OP2	OP2	OP3	OP2
3 BYTE	1	1	0	0	OP1	OP2	OP3	OP0
	1	1	0	1	OP1	OP1	OP2	OP3
	1	1	1	0	OP1	OP2	OP1	OP2
	1	1	1	1	OP1	OP1	OP2	OP1
LONG WORD	0	0	0	0	OP0	OP1	OP2	OP3
	0	0	0	1	OP0	OP0	OP1	OP2
	0	0	1	0	OP0	OP1	OP0	OP1
	0	0	1	1	OP0	OP0	OP1	OP0

The operand representation and size/offset encodings for external data bus multiplexing

are shown in Figure 24.



TRANSFER SIZE	SIZ1	SIZ0
BYTE	0	1
WORD	1	0
3 BYTE	1	1
LONG WORD	0	0

OFFSET	A1	A0
BYTE	0	0
WORD	0	1
3 BYTE	1	0
LONG WORD	1	1

SIZE AND OFFSET ENCODING FOR EXTERNAL DATA BUS MULTIPLEXING

Figure 24 Operand representation and size/offset encodings

The following are the examples of long word transfers to a 16-bit, and to an 8-bit data bus.

BUS CYCLE	SIZ1	SIZ0	A1	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	D31 DATA BUS D16	
1	0	0	0	0	0	1	OP0	OP1
2	1	0	1	0	0	1	OP2	OP3

Figure 25 Long word transfer to 16-bit data bus

BUS CYCLE	SIZ1	SIZ0	A1	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	D31 DATA BUS D24
1	0	0	0	0	1	0	OP0
2	1	1	0	1	1	0	OP1
3	1	0	1	0	1	0	OP2
4	0	1	1	1	1	0	OP3

Figure 26 Long word transfer to 8-bit data bus

An address error exception occurs when an instruction fetch at an odd address is attempted, although no restriction is imposed on data alignment. The next two figures shows the misaligned longword/word transfers to 32/16 bit buses, respectively.

BUS CYCLE	SIZ1	SIZ0	A2	A1	A0	$\overline{DSACK1}$	$\overline{DSACK0}$	D31	DATA BUS		D0
1	0	0	0	1	1	0	0	XXX	XXX	XXX	OP0
2	1	1	1	0	0	0	0	OP1	OP2	OP3	XXX

Figure 27 Misaligned longword transfer to 32-bit data bus

BUS CYCLE	SIZ1	SIZ0	A2	A1	A0	$\overline{DSACK1}$	$\overline{DSACK0}$	D31	DATA BUS	D16
1	1	0	0	0	1	0	1	XXX	OP2	
2	0	1	0	1	0	0	1	OP3	XXX	

Figure 28 Misaligned word transfer to 16-bit data bus

2. Bus Operation

- Read Cycle: Data is received from external device in accordance with the following sequence of events:

MC68020	**External Device**
Sets Read/Write to Read	
Puts Address onto address bus	
Drives Size outputs	
Asserts External Cycle Start/ Operand Cycle Start	
Asserts Address Strobe	
Asserts Data Strobe	
Asserts Data Buffer Enable	
	Decodes address
	Puts data onto data bus
	Asserts Data Transfer and Size Acknowledge
Latches data	
Negates Data Strobe	
Negates Address Strobe	
Negates Data Buffer Enable	

Removes Data from the bus
Negates Data Transfer and
Size Acknowledge

Starts new bus cycle

- Write Cycle: Data is sent to external device in accordance with the following sequence of events:

MC68020 **External Device**

Sets Read/Write to Write
Drives Function Codes
Puts Address onto address bus
Drives Size outputs
Asserts External Cycle Start/
Operand Cycle Start
Asserts Address Strobe
Asserts Data Strobe
Asserts Data Buffer Enable

Decodes address
Latches data from data bus
Asserts Data Transfer and
Size Acknowledge

Negates Data Strobe
Negates Address Strobe
Removes Data from data bus
Negates Data Buffer Enable

Negates Data Transfer and
Size Acknowledge

Starts new bus cycle

- Read-Modify-Write Cycle: During this cycle, data is read from memory, it is modified in ALU and written back to the same address. This bus cycle is indivisible, that is, MC68020 does not release the bus until the whole cycle is completed. This feature is utilized in multi processesing by the instructions Test And Set (TAS) and Compare And Swap (CAS, CAS2). For detailed explanation, refer to Reference 2.

3. Interrupt Operation

MC68020 has seven interrupt levels of which level seven is the highest. The level of requested interrupt is signalled to the processor via interrupt priority level signals IPL2-IPL0. Level zero (IPL2-IPL0 = HHH) means no interrupt requested. If the level of requested interrupt is between one and six, the interrupt level is compared against the interrupt mask level in the status register. If the requested interrupt level is less than or equal to the mask level, the interrupt is ignored. Otherwise the interrupt is processed. The level seven interrupts are non-maskable; that is, they are immediately processed regardless of the interrupt mask level in the status register.

The following two rules guarantee the processing of an interrupt:

- Except for the level seven interrupt, the interrupt level should be higher than the interrupt mask level in the status register.
- IPL0 through IPL2 should stay at the requested level, until the interrupt is acknowledged by MC68020.

It is also possible that an interrupt request of a duration as short as two clock cycles can be processed. A recognized interrupt is made pending and is processed at the next instruction boundary, unless a higher level interrupt is valid. After the interrupt is made pending, the processor first determines the starting location of the interrupt handling routine pointed by the interrupt vector number. This vector number can be generated internally or can be provided by the interrupt requesting device through the data bus in the interrupt acknowledge cycle.

The following is the flowchart for Interrupt Acknowledge Sequence;

- Interrupt Acknowledge Sequence

MC68020	**DEVICE**
	Requests interrupt
Compares the requested interrupt level with the mask level. Sets Read/Write output to Read Sets Function Code to 7 (CPU Space) Sets A1-A3 to the recognized level. Sets Size outputs to Byte. Asserts Address Strobe and Data Strobe	
	Either Places vector number on data bus. Asserts Data Transfer and Size Acknowledge.

Or
Asserts !AVEC for internal generation
of interrupt vector number.

Gets the interrupt vector number.
Negates Address Strobe and Data Strobe.

Negates Data Transfer and
Size Acknowledge, if asserted.

Processes the interrupt.

In case of a spurious interrupt, that is, an interrupt request is recognized, but !DSACKX or !AVEC signal is not asserted by the external device, then the external circuit should assert !BERR signal. This terminates the interrupt vector acqusition and causes MC68020 to fetch spurious interrupt vector and to start exception processing.

4. Breakpoint Acknowledge Cycle

This cycle is initiated by the execution of a breakpoint instruction during which MC68020 reads a word from CPU space. Upon the termination of the cycle by !DSACKx, the processor replaces the breakpoint instruction by the data read from the data bus and continue to execute that instruction. If the cycle is terminated by !BERR, then the processor continues with processing an illegal instruction exception.

5. Coprocessor Operations

MC68020 communicates with the coprocessor by performing CPU space accesses. During a CPU access, address bus contains the access information, instead of an address. The lines A16 through A19 contain 0010 to specify coprocessor operation, and the coprocessor ID number to be accessed is encoded on the lines A13 through A15. The lines A0 through A5 indicate the coprocessor interface register to be accessed. The coprocessor ID number 0 is belong to MC68020 memory management unit.

6. Bus Error Operation

MC68020 is provided with a Bus Error input which is used to terminate the current bus cycle, in case of a handshake failure. The signal for this input should be generated externally, after the maximum time period between the assertion of !AS and !DSACKx. Bus error input is also used to suspend the execution of an instruction, if an invalid memory access is detected.

MC68020 may start to process the bus error exception immediately, in case of a data space access, or may defer processing it, if the bus error occurs during an instruction prefetch. In the second case, the bus error exception will occur, when the faulted data is actually to be executed.

7. Retry Operation

If both !BERR and !HALT inputs are asserted externally, MC68020 will rerun the previous bus cycle after the negation of these two signals. There is no restriction on the type of bus cycle to be retried.

8. Halt Operation

MC68020 will stop all external bus activity when the !HALT input is asserted. The internal operation of the processor is not affected by the !HALT input. For example, a program stored in the cache memory will continue to run regardless of the !HALT input. Stepping through the processor operation one bus cycle at a time is also possible by asserting the !HALT input when the processor starts a bus cycle. As long as the !HALT input remains asserted, the current bus cycle will be completed, but the next cycle will not start. In order to step through the next bus cycle, the !HALT input should be negated and then asserted again after the bus cycle starts.

9. Double Bus Fault

Double bus fault is an address or bus error which occurs during the exception processing for an address error, bus error, or reset exception. When a double bus fault occurs, the processor halts and the !HALT line is asserted. Then the processor can only be started by an external reset.

10. Reset Operation

The reset operation is bidirectional, the processor can reset the external devices, or the external circuitry can reset the processor. In order to reset the processor, the !RESET input should be asserted at least 100 ms. Then the processor loads the interrupt stack pointer and the program counter from the long-word addresses $00000000 and $00000004 respectively. Trace is disabled, privileged states is set to supervisor-interrupt state by clearing/setting the relevant bits in the status register. The vector base register is set to $00000000 and the cache is disabled by clearing the cache enable bit in the cache control register. The other registers remain unaffected.

The processor resets the external devices by executing a RESET instruction, which asserts the !RESET line for 512 clock cycles. Nothing inside the processor is affected by executing the RESET instruction.

APPENDIX D: PROCESSING STATES OF MC68020

This section describes the operation of the processor in two subsections, the privilege states and the exception processing.

1. Privilege States

MC68020 has two levels of privilege, the supervisor level and the user level. The supervisor level has a higher privilege than the user level in that the user level is not allowed to access all the program and data areas and to execute all the instructions. This separation of privileges provides security in the system.

a. Supervisor States

The S bit in the status register determines the privilege level of the processor. When the S bit is set, the processor runs in supervisor state and can execute all the instructions. The M bit in the status register allows the separation of the supervisor stack for user and interrupt-associated tasks. This separation increases efficiency in multi-tasking environment. When the M bit is set, the system stack pointer references the master stack pointer, otherwise the interrupt stack pointer is used as the system stack pointer. Referencing the system stack pointer is the only operation affected by the status of the M bit. After reset, the S bit is set and the M bit is cleared. If the M bit is already set and an interrupt occurs, then the processor saves the status of the M bit and clears it to process the exception for interrupt. When processor runs in the supervisor state, the S and M bits can be manipulated by the instructions that modify the status register. The supervisor state is encoded as 5 (data) and 6 (program) on the function code pins. By executing the instructions RTE, MOVE to SR, ANDI to SR and EORI to SR, the processor can switch from the supervisor state to the user state.

b. User State

When the S bit in the status register is set to zero, MC68020 runs in the user state in which the instructions that have an impact on the system are not allowed to execute. In the user state, the system stack pointer references the user stack pointer. The user

state is encoded as 1 (data) and 2 (program) on the function code pins. The exception processing is the only way to switch from the user state to the supervisor state.

2. Exception Processing

a. General Information

An exception can be generated internally by instructions, address errors, tracing or breakpoints; it can also be generated externally by interrupts, bus errors, reset or errors detected by coprocessor. The following are the four steps to process an exception as explained in the section "MC68020 Overview":

- Make an internal copy and set/clear the required bits of the status register for exception processing.
- Determine the exception vector.
- Save the current processor context on the active supervisor stack.
- Get the new processor context and proceed with the instruction processing.

The internal copy of the status register is saved on the exception stack frame created in order to save the current processor context. Depending on the type of the exception, MC68020 can create exception stack frames in six different formats. All of the six frames have at least four fields that contain

- Status Register
- Program Counter
- Format of the frame
- Vector Offset

Some exception stack frames have another field which contains additional processor information. This information can be 2, 6, 12 or 42 words in length. Detailed information on the exception stack frames can be found at the end of this appendix.

After saving the current content of status register, the processor is switched to the supervisor state by setting the S bit. The trace bits are cleared in order to prevent the exception handler from being hindered by tracing.

In the second step, the MC68020 determines the exception vector number. The vector number is obtained by a read from CPU space for interrupts (if the interrupt is not autovectored). The coprocessor provides the vector number in exception primitive response, if it detects an exception. The vector numbers for all other exceptions are generated internally.

In the third step, if the exception is not reset, an exception stack frame is created on the active supervisor stack and the current processor context is saved in this frame. With the M bit set, if the exception is an interrupt, then the MC68020 clears the M bit and creates another stack frame on the interrupt stack.

In the last step, the exception vector offset is calculated by multiplying the exception vector number by four (number of bytes in a long-word). The calculated exception vector offset is then added to the contents of the vector base register (default value after reset is 00000000 Hex) to locate the exception vector address. The contents of the exception vector address is loaded into the program counter (for reset exception, the interrupt stack pointer is also loaded from the exception vector address) and the instruction at the address pointed by the program counter is fetched and the instruction execution resumes. All the exceptions are grouped and are given priorities to determine the order in which simultaneous exceptions will be handled. The exception groups and the level of priorities are as follows;

- Group 0: Priority 0 Reset

- Group 1: Priority 1 Address Error
  Priority 2 Bus Error

- Group 2: Priority 3 BKPT #n
  CALLM
  CHK
  CHK2
  cp Mid-Instruction
  cp Protocol Violation
  cp TRAPcc
  Divide-By-Zero
  RTE
  RTM
  TRAP #n
  TRAPV

- Group 3: Priority 4 Illegal instruction
  Line A
  Unimplemented Line F
  Privilege violation
  cp Pre-Instruction
- Group 4: Priority 5 cp Post-Instruction
  Priority 6 Trace
  Priority 7 Interrupt

b. The sources of exceptions

(1) Reset

This is the highest priority exception which initializes the system and recovers the system from a catastrophic failure. The current process can not be recovered after a reset. When an external reset signal is applied to the !RESET input, MC68020 takes the following steps;

- The status register:
  Trace bits T0, T1 are cleared (tracing disabled).
  S bit is set, M bit is cleared (supervisor interrupt state).
  Interrupt mask level is set to level seven.
- The vector base register:
  is initialized to 00000000 Hex.
- The cache control register:
  is initialized to 00000000 Hex.
- The vector number:
  is internally generated to point the reset exception vector at zero offset in the supervisor program space. The length of reset exception vector is two long words, the first of which holds the initial value for interrupt stack pointer and the second the initial value for the program counter.
- Program execution starts with the instruction fetched from the address pointed by the program counter.

When a RESET instruction is executed, no internal registers of MC68020 are affected, only the !RESET line is asserted for 512 clock cycle to reset the external devices. The program execution continues with the next instruction.

(2) Address Error

When an attempt is made to fetch an instruction from an odd address, then the address error exception occurs, and the bus cycle is not executed. If the occurance of

the address error coincides with the processing of a bus error, address error or reset exception, then the processor halts.

(3) Bus Error

When the !BERR input is asserted by the external logic during a bus cycle, then the current bus cycle is aborted. The exception processing begins immediately if the aborted bus cycle is a data space access. The processor defers the exception processing until the prefetched instruction is actually needed, if the aborted cycle is an instruction prefetch. Depending on when the bus error occurs during a bus cycle, MC68020 creates one of two exception stack frames for the bus error. If the bus error occurs in the middle of instruction execution, then the larger stack frame (Format B Hex) is required, otherwise exception stack frame in Format A Hex is created. As in the address error, if the bus error takes place during the exception processing for an address error, bus error, reset, or RTE instruction execution, the MC68020 halts.

(4) Instruction Trap

The detection of an abnormal condition during instruction execution or executing some specific instructions cause a trap. The exception vector number is generated internally for all instruction traps (the TRAP #n instruction has part of the vector number in itself). The instructions that specifically generate a trap are as follows:

- TRAP #n : When executed, forces an exception. By using this instruction, user programs can make system calls.
- TRAPcc, TRAPV, cpTRAPcc, CHK, CHK2 : An exception is forced by these instructions, if the user program detects a run-time error.
- DIVS, DIVU : If a division operation with a zero divisor is attempted, these two instructions generate a exception.

(5) Breakpoint

Unlike the MC68000 and MC68008, inserting an illegal instruction into the breakpoint address and looking for a fetch from an illegal instruction exception vector address is not a reliable way to determine if the breakpoint has been reached, in a MC68020 system. This is due to the allowance of multiple exception vector tables by using the vector base register. Instead, the opcodes 4848 Hex through 484F Hex are used as breakpoint instructions. By using breakpoints, MC68020 can be used in a hardware emulator, and the execution of a program in the on-chip cache memory can

be monitored by external hardware.

(6) Format Error

The MC68020 checks the format of control data, as well as the validity of the prefetched instruction. The control data checked by the processor include the option and type fields in the module descriptor for CALLM and RTM, the format of the stack for RTE and RTM, the format of the coprocessor save area for cpRESTORE. If the format check of the control data fails, then the MC68020 generates a format error exception, and creates a short format frame (Format 0 Hex). The program counter value saved on the satck frame points to the address of the instruction that detected the format error.

(7) Illegal or Unimplemented Instruction

Any word bit pattern that does not match with the bit pattern of the first word of a legal MC68020 instruction is called **illegal instruction**. Illegal instructions also include the MOVEC instruction, if it has an undefined register specification in the first extension word. There are two types of unimplemented instructions, **A-line opcodes and F-line opcodes**, where A and F correspond to the numbers that bits 15 through 12 of the opcode represent in hexadecimal form. F-line opcodes are used for coprocessor instructions. Illegal instructions and unimplemented instructions have distinct exception vectors which allows the emulation of unimplemented instructions more efficiently.

(8) Privilege Violation

An attempt to execute one of the following instructions in the user privilege state will cause an exception;

- ANDI to SR
  EORI to SR
  ORI to SR
  MOVE to SR
  MOVE from SR
  MOVE USP
  MOVEC
  MOVES
  RESET
  RTE
  cpSAVE
  cpRESTORE
  STOP

Also it is possible that an exception will occur when the coprocessor requests a privilege check, while MC68020 is in the user state.

Both the next instruction address and the address of the instruction that caused privilege violation are saved on the exception stack frame.

(9) Tracing

By setting the trace bits in the status register, programs can be traced on instruction-by-instruction basis. The MC68020 can also trace the instructions that change the sequential flow of the program. The trace bits indicate the type of tracing as shown in Table 10:

Table 10 Trace Bit Encoding.

T1	T0	TRACE
0	0	NO TRACE
0	1	TRACE BRANCH
1	0	TRACE ALL
1	1	UNDEFINED

Tracing allows a debugger program, like the one written in Reference 1, to monitor the execution of a test program.

In no trace mode, the instructions are executed normally. When the trace bits are set to trace branch mode, the instructions that change the sequential flow of the program will be traced. These instructions include all branches, jumps, instruction traps, returns and those that affect the status register contents. If trace bits are set to trace all mode, every instruction will be traced. The exception processing for a trace starts after the completion of the traced instruction and before the execution of next instruction. For trace exception processing, MC68020 creates a stack frame in Format 2 Hex and clears the trace bits. Both the address of the next instruction and the address of the traced instruction are saved on the stack frame. If the STOP instruction begins the execution, when T1 bit is set, then the stop instruction will not take effect.

(10) Interrupts

The interrupt mask level in the status register determines whether an interrupt will be processed or ignored. If the requested interrupt has a higher priority level than the interrupt mask level, then the interrupt is made pending and the processing begins at the next instruction boundary, otherwise the interrupt is ignored. The level seven

interrupt is an exception to this case, it can not be inhibited by the interrupt mask level.

During an interrupt acknowledge cycle, the level of the interrupt being acknowledged is put on the address lines A1-A3, and if the interrupt is not autovectored, the vector number is fetched from the external device. If the interrupt is autovectored, the MC68020 internally generates a vector number which corresponds to the level of the interrupt. If a bus error is detected, then the spurious interrupt vector is fetched.

(11) Return From Exception

The Return From Exception (RTE) instruction is used to return to the processor context prior to the exception, whenever it is possible. The processor examines the stack frame created for the exception in order to check the validity of the frame and to determine the type of context restoration. In case of a format or bus error during the execution of the RTE instruction, another stack frame is created above the frame which was going to be used.

c. Exception Stack Frames

Depending on the type of the exception, the MC68020 creates one of six stack frames which are described in this section.

(1) Normal Four Word Stack Frame (Format $0)

- Created by

  Interrupts
  Format Errors
  TRAP #n Instructions
  Illegal and Unimplemented Instructions
  Privilege Violations
  Coprocessor pre-instruction Exceptions

- The format of the frame (see Figure 29);

  SP = Status Register
  SP + 02 Hex = Program Counter
  SP + 06 Hex = Format Number (0000 Hex) + Vector Offset (12 Bits)

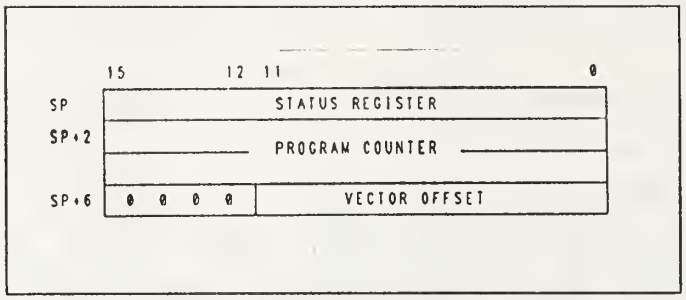


Figure 29 Normal four-word stack frame

- The program counter value (SP+02 Hex) is the address of the instruction that caused the exception or the address of the next instruction.

(2) Throwaway Four-Word Stack Frame (Format $1)

- Created if the supervisor state is changed to interrupt state from master state (M bit is cleared) during exception processing for an interrupt.
- The format of the frame (see Figure 30):

SP = Status Register
SP + 02 Hex = Program Counter
SP + 06 Hex = Format Number (0001 Hex) + Vector Offset (12 Bits)

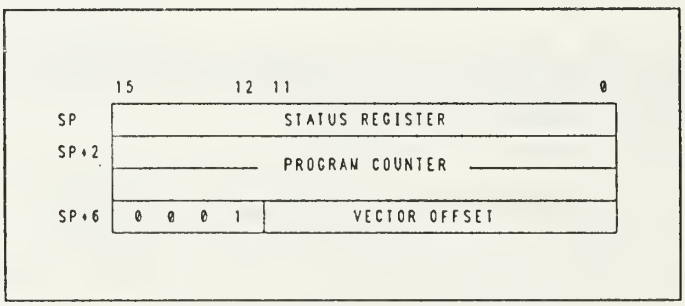


Figure 30 Throwaway four-word stack frame

- The program counter value (SP+02 Hex) might be the address of the instruction that caused the exception, the address of the next instruction, or coprocessor mid-instruction stack frame.

(3) Normal Six Word Stack Frame (Format $2)

- Created by

Coprocessor post-instruction exceptions
CHK and CHK2 instructions
cpTRAPcc, TRAPcc and TRAPV instructions
Trace

Zero divide

- The format of the frame (see Figure 31):

  SP = Status Register
  SP + 02 Hex = Program Counter
  SP + 06 Hex = Format Number (0001 Hex) + Vector Offset (12 Bits)
  SP + 08 Hex = Instruction Address (32 Bits)

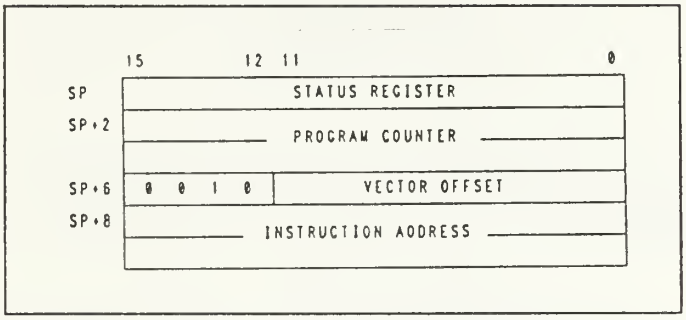


Figure 31 Normal six-word stack frame

- The program counter value (SP+02 Hex) is the address of the next instruction, or the address to be returned by RTE instruction.
- The instruction address value is the address of the instruction that caused the exception.

(4) Coprocessor Mid-instruction Exception Stack Frame (Format $9)

- Created when

  "Take mid-instruction exception" coprocessor primitive is read while the MC68020 is processing a coprocessor instruction.

  The MC68020 detects a protocol violation during a coprocessor instruction processing.

  "Null, come again with interrupts allowed" primitive is read, and the MC68020 detects a pending interrupt.

- The format of the frame (see Figure 32):

  SP = Status Register
  SP + 02 Hex = Program Counter
  SP + 06 Hex = Format Number (0010 Hex) + Vector Offset (12 Bits)
  SP + 08 Hex = Instruction Address (32 Bits)
  SP + 0C Hex = Internal Registers (4 Words)

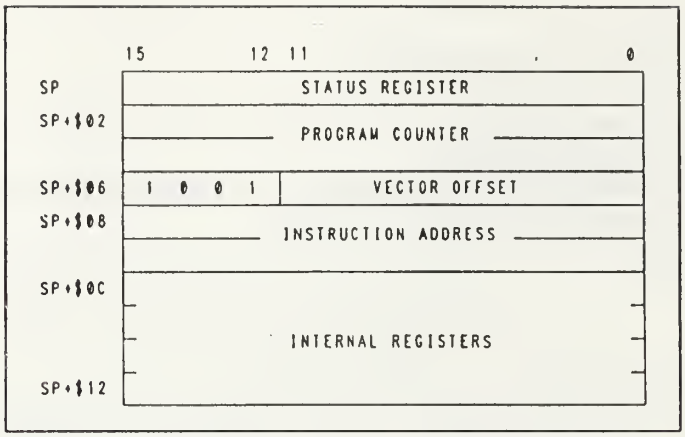


Figure 32 Coprocessor mid-instruction exception stack frame

- The program counter value (SP+02 Hex) is the address of the next instruction.
- The instruction address value is the address of the instruction that caused the exception.

(5) Short Bus Cycle Stack Frame (Format $A)

- Created when

  the MC68020 detects a bus cycle fault, and recognizes it is at an instruction boundary.

- The format of the frame (see Figure 33):

  SP = Status Register
  SP + 02 Hex = Program Counter
  SP + 06 Hex = Format Number (0101 Hex) + Vector Offset (12 Bits)
  SP + 08 Hex = Internal Register
  SP + 0A Hex = Special Status Word
  SP + 0C Hex = Instruction Pipe Stage C
  SP + 0E Hex = Instruction Pipe Stage B
  SP + 10 Hex = Data Cycle Fault Address
  SP + 14 Hex = Internal Register
  SP + 16 Hex = Internal Register
  SP + 18 Hex = Data Output Buffer
  SP + 1C Hex = Internal Register
  SP + 1E Hex = Internal Register

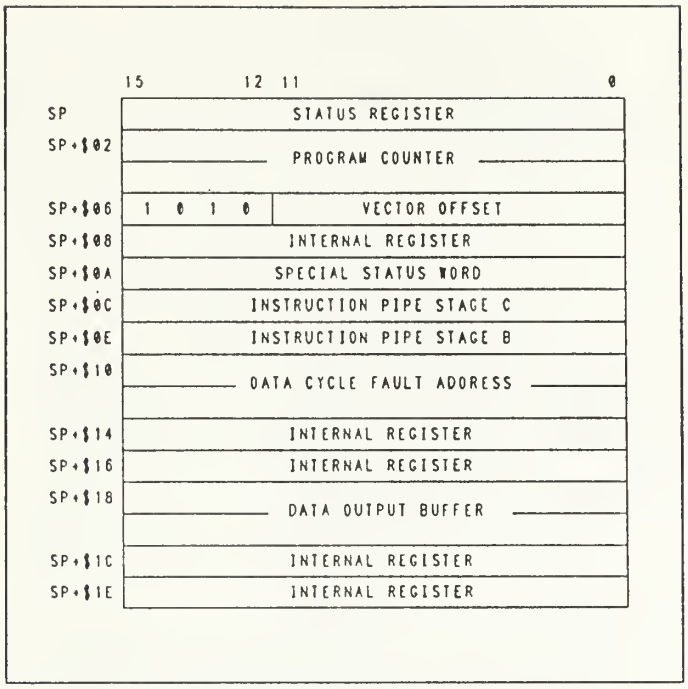


Figure 33 Short bus cycle fault stack frame

- The program counter value (SP+02 Hex) is the address of the next instruction.

(6) Long Bus Cycle Stack Frame (Format B Hex)

- Created when

  the MC68020 detects a bus cycle fault, and recognizes it is not at an instruction boundary.

- The format of the frame (see Figure 34):

SP = Status Register (Word)
SP + 02 Hex = Program Counter (2 Words)
SP + 06 Hex = Format Number (0101 Hex) + Vector Offset (12 Bits)
SP + 08 Hex = Internal Register (Word)
SP + 0A Hex = Special Status Word (Word)
SP + 0C Hex = Instruction Pipe Stage C (Word)
SP + 0E Hex = Instruction Pipe Stage B (Word)
SP + 10 Hex = Data Cycle Fault Address (2 Words)
SP + 14 Hex = Internal Registers (2 Words)
SP + 18 Hex = Data Output Buffer (2 Words)
SP + 1C Hex = Internal Registers (4 Words)
SP + 24 Hex = Stage B Address (2 Words)
SP + 28 Hex = Internal Registers (2 Words)
SP + 2C Hex = Data Input Buffer (2 Words)
SP + 30 Hex = Internal Registers (22 Words)

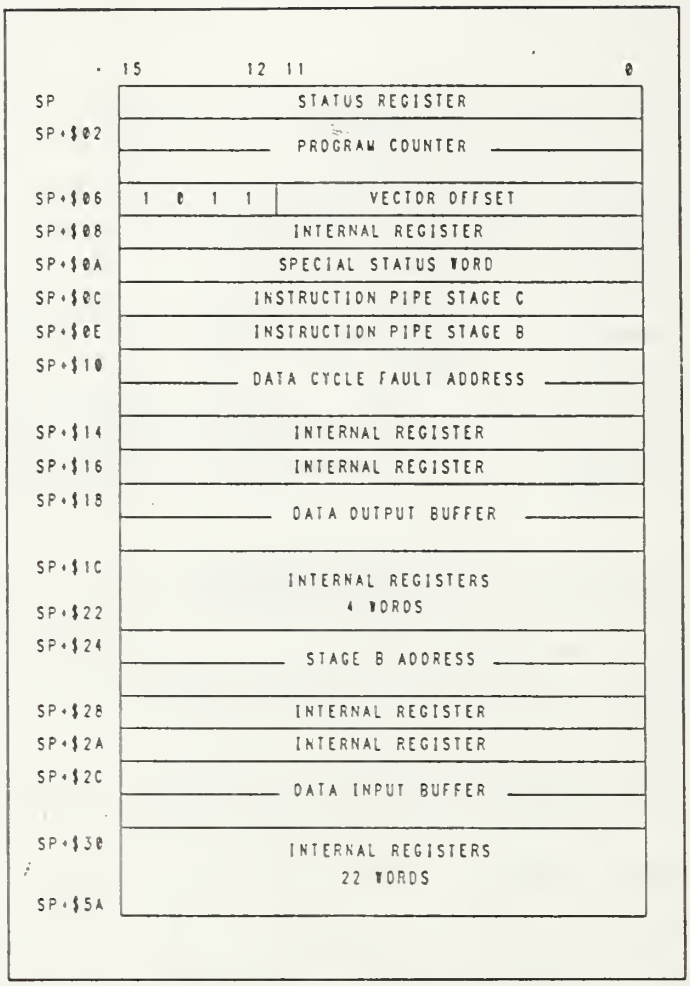


Figure 34 Long bus cycle fault stack frame

- The program counter value (SP+02 Hex) is the address of the instruction that was executing when the bus cycle fault occured (not necessarily the instruction that caused the bus error).

d. Coprocessor-related exceptions

These exceptions can be divided in two groups, coprocessor-detected exceptions and main processor-detected exceptions. The main difference between two groups is the point at which the exception processing starts. Due to concurrent instruction execution, the processing for many of the coprocessor-detected exceptions does not start until the main processor completes the execution of the offending instruction and attempts to execute the next instruction. The exception processing for all main processor detected exceptions and some coprocessor-detected exception starts during the execution of the offending instruction.

(1) The Coprocessor-Detected Exceptions.

The coprocessor-detected exceptions can be either related to the communication with the main processor or to the execution of a floating-point instruction. The exception vector numbers and address offsets for coprocessor-related exceptions are as follows:

Vector Number	Vector Offset (Hex)	Assignment
7	1C	FTRAPcc instruction
11	2C	F-Line emulator
13	34	Coprocessor Protocol Violation
48	C0	Branch or Set on Unordred Condition
49	C4	Inexact Result
50	C8	Floating-point divide by zero
51	CC	Underflow
52	D0	Operand Error
53	D4	Overflow
54	D8	Signalling Not-A-Number

The execution of a floating-point instruction can cause one or more of eight exceptions. The exceptions caused by the instruction "move floating-point data register to an external location" are called mid-instruction exceptions. All the other instruction exceptions are pre-instruction exceptions.

- Signalling Not-A-Number: The data types defined by the user or non-IEEE data types cause SNAN exception. This exception is never caused as a result of an operation. The instructions that do not modify the status bits must be used in SNAN trap handler to hinder further exceptions.
- Operand Error: If the current operation has no mathematical interpretation for the given operands, then an operand error occurs.
- Overflow: When the exponent of the result is greater than or equal to the maximum value for the specified format, then overflow condition can be detected. But the exception occurs if the destination is in one of the floating-point formats. Overflows for destinations in integer or packed decimal format, are included as operand errors.
- Underflow: When the exponent of the result is less than or equal to the minimum value for the specified format, then overflow condition can be detected. But the exception occurs if the destination is in one of the floating-point formats. Overflows for destinations in integer or packed decimal format, are included as

operand errors.

- Divide-By-Zero: A division with zero divider or a transcendental function which is asymptotic with infinity will cause Divide-By Zero exception.
- Inexact Result (INEX2): This exception will occur, if the result of an operation, except for an operation with packed decimal operand, has a mantissa that can not be represented in the specified rounding precision or the destination precision.
- Inexact Result (INEX1): This exception will occur, if the result of an operation with packed decimal operand, has a mantissa that can not be represented in the specified rounding precision or the destination precision.
- Branch/Set on Unordered: The conditional instructions with the following IEEE non-aware branch condition predicates can cause BSUN exception.

Table 11 IEEE non-aware branch condition predicates

GT	GREATER THAN
NGT	NOT GREATER THAN
GE	GREATER THAN OR EQUAL
NGE	NOT GREATER THAN OR EQUAL
LT	LESS THAN
NLT	NOT LESS THAN
LE	LESS THAN OR EQUAL
NLE	NOT LESS THAN OR EQUAL
GL	GREATER OR LESS THAN
NGL	NOT GREATER OR LESS THAN
GLE	GREATER OR LESS OR EQUAL
NGLE	NOT GREATER OR LESS OR EQUAL
SF	SIGNALLING TRUE
ST	SIGNALLING FALSE
SEQ	SIGNALLING EQUAL
SNE	SIGNALLING NOT EQUAL

(2) Coprocessor Detected Protocol Violations

A protocol violation occurs, when the command, condition, register select or operand CIR is accessed unexpectedly as follows:

- When a write to the command or condition CIR is expected, but the register select or operand CIR is accessed.
- When a read from the register select or operand CIR is expected, but a write to the command, condition or operand CIR occurs.
- When a write to the operand CIR is expected, but either a write to the command

or condition CIR or a read from the register select or operand CIR occurs.After detecting a protocol violation, the MC68881 encodes the response CIR with the take pre-instruction primitive so that the MC68020 will terminate the dialog.

Table 12 MC68020 Exception Vector Table.

VECTOR NUMBER (DECIMAL)	VECTOR OFFSET (HEX)	VECTOR ASSIGNMENT
0	000	RESET : INITIAL ISP
1	004	RESET : INITIAL PC
2	008	BUS ERROR
3	00C	ADDRESS ERROR
4	010	ILLEGAL INSTRUCTION
5	014	ZERO DIVIDE
6	018	CHK,CHK2 INSTRUCTION
7	01C	cpTRAPcc, TRAPcc, TRAPV INSTRUCTIONS
8	020	PRIVILEGE VIOLATION
9	024	TRACE
10	028	LINE 1010 EMULATOR
11	02C	LINE 1111 EMULATOR
11	030	UNASSIGNED
13	034	COPROCESSOR PROTOCOL VIOLATION
14	038	FORMAT ERROR
15	03C	UNINITIALIZED INTERRUPT
16 through 23	040 through 05C	UNASSIGNED
24	060	SPURIOUS INTERRUPT
25	064	LEVEL 1 INTERRUPT AUTOVECTOR
26	068	LEVEL 2 INTERRUPT AUTOVECTOR
27	06C	LEVEL 3 INTERRUPT AUTOVECTOR
28	070	LEVEL 4 INTERRUPT AUTOVECTOR
29	074	LEVEL 5 INTERRUPT AUTOVECTOR
30	078	LEVEL 6 INTERRUPT AUTOVECTOR
31	07C	LEVEL 7 INTERRUPT AUTOVECTOR
32 through 47	080 through 0BC	TRAP #0 - 15 INSTRUCTION VECTORS
48	0C0	FPCP BSUN
49	0C4	FPCP INEX
50	0C8	FPCP DZ
51	0CC	FPCP UNFL
52	0D0	FPCP OPERR
53	D40	FPCP OVFL
54	0D8	FPCP SNAN
55	0DC	UNASSIGNED
56	0E0	PMMU CONFIGURATION
57	0E4	PMMU ILLEGAL OPERATION
58	0E8	PMMU ACCESS LEVEL VIOLATION
59 through 63	0EC through 0FC	UNASSIGNED
64 through 255	100 through 3FC	USER DEFINED

Table 13 MC68020 Extensions To M68000 Family Instructions

INSTRUCTION	EXTENSION
Bcc	32 BIT DISPLACEMENT
Bfxxxx	BIT FIELD INSTRUCTIONS
BKPT	NEW INSTRUCTION
BRA	32 BIT DISPLACEMENT
BSR	32 BIT DISPLACEMENT
CALLM	NEW INSTRUCTION
CAS, CAS2	NEW INSTRUCTION
CHK	32 BIT OPERANDS
CHK2	NEW INSTRUCTION
CMPI	PC RELATIVE ADDRESSING MODE
CMP2	NEW INSTRUCTION
cp	COPROCESSOR INSTRUCTIONS
DIVS/DIVU	32 BIT AND 64 BIT OPERANDS
EXTB	8 BIT EXTEND TO 32 DITS
LINK	32 BIT DISPLACEMENT
MOVEC	NEW CONTROL REGISTERS
MULS/MULU	32 BIT OPERANDS
PACK	NEW INSTRUCTION
RTM	NEW INSTRUCTION
TST	PC RELATIVE ADDRESSING MODE
TRAPcc	NEW INSTRUCTION
UNPK	NEW INSTRUCTION

Table 14 MC68020's Improved Features.

FEATURE	IMPROVEMENT
DATA BUS	8, 16 OR 32 BITS (DYNAMIC SIZING)
ADDRESS BUS	32 BITS
INSTRUCTION CACHE	128 WORDS
COPROCESSOR INTERFACE	IMPLEMENTED IN MICROCODE
DATA ALIGNMENT	ONLY INSTRUCTIONS WORD ALIGNED
CONTROL REGISTERS	SFC, DFC, VBR, CACR, CAAR
STACK POINTERS	USP, SSP (ISP and MSP)
STATUS REGISTER	T0/T1, S, M, I MASK, COND. CODE
ADDRESS SPACE	CPU SPACE = FUNCTION CODE 7
STACK FRAMES	$0, $1, $2, $9, $A, $B

Table 15 MC68020 Instruction Set.

INSTRUCTION	
ABCD	ADD DECIMAL WITH EXTEND
ADD	ADD
ADDA	ADD ADDRESS
ADDI	ADD IMMEDIATE
ADDQ	ADD QUICK
ADDX	ADD WITH EXTEND
AND	LOGICAL AND
ANDI	LOGICAL AND IMMEDIATE
ASL, ASR	ARITHMETIC SHIFT LEFT,RIGHT
Bcc	BRANCH CONDITIONALLY
BCHG	TEST BIT AND CHANGE
BCLR	TEST BIT AND CLEAR
BFCHG	TEST BIT FIELD AND CHANGE
BFCLR	TEST BIT FIELD AND CLEAR
BFEXTS	SIGNED BIT FIELD EXTRACT
BFEXTU	UNSIGNED BIT FIELD EXTRACT
BFFFO	BIT FIELD FIND FIRST ONE
BFINS	BIT FILED INSERT
BFSET	TEST BIT FIELD AND SET
BFTST	TEST BIT FIELD
BKPT	BREAKPOINT
BRA	BRANCH
BSET	TEST BIT AND SET
BSR	BRANCH TO SUBROUTINE
BTST	TEST BIT
CALLM	CALL MODULE
CAS	COMPARE AND SWAP OPERANDS
CAS2	COMPARE AND SWAP DUAL OPER.
CHK	CHECK REG. AGAINST BOUND
CHK2	CHK REG AGST UP/LOW BOUUNDS
CLR	CLEAR
CMP	COMPARE
CMPA	COMPARE ADDRESS
CMPI	COMPARE IMMMEDIATE

INSTRUCTION	
CMPM	COMPARE MEMORY TO MEMORY
CMP2	CMP REG AGST UP/LOW BOUNDS
DBcc	TEST COND. DECR. AND BRANCH
DIVS,DIVSL	SIGNED DIVIDE
DIVU,DIVUL	UNSIGNED DIVIDE
EOR	LOGICAL EXCLUSIVE OR
EORI	LOGICAL EXCLUSIVE OR IMMED
EXG	EXCHANGE REGISTERS
EXT,EXTB	SIGNED EXTEND
ILLEGAL	TAKE ILLEGAL INSTRUC. TRAP
JMP	JUMP
JSR	JUMP TO SUBROUTINE
LEA	LOAD EFFECTIVE ADDRESS
LINK	LINK AND ALLOCATE
LSL,LSR	LOGICAL SHIFT LEFT, RIGHT
MOVE	MOVE
MOVEA	MOVE ADDRESS
MOVE CCR	MOVE CONDITION CODE REGIST.
MOVE SR	MOVE STATUS REGISTER
MOVE USP	MOVE USER STACK POINTER
MOVEC	MOVE CONTROL REGISTER
MOVEM	MOVE MULTIPLE REGISTERS
MOVEP	MOVE PERIPHERAL
MOVEQ	MOVE QUICK
MOVES	MOVE ALTERNATE ADR. SPACE
MULS	SIGNED MULTIPLY
MULU	UNSIGNED MULTIPLY
NBCD	NEGATE DECIMAL WITH EXTEND
NEG	NEGATE
NEGX	NEGATE WITH EXTEND
NOP	NO OPERATION
NOT	LOGICAL COMPLEMENT
OR	LOGICAL INCLUSIVE OR
ORI	LOGICAL INCLUSIVE OR IMMED

INSTRUCTION	
PACK	PACK BCD
PEA	PUSH EFFECTIVE ADDRESS
RESET	RESET EXTERNAL DEVICES
ROL, ROR	ROTATE LEFT, RIGHT
ROXL,ROXR	ROTATE WITH EXTEND LEFT,RIG
RTD	RETURN AND DEALLOCATE
RTE	RETURN FROM EXCEPTION
RTM	RETURN FROM MODULE
RTR	RETURN AND RESTORE CODES
RTS	RETURN FROM SUBROUTINE
SBCD	SUBTRACT DECIMAL WITH EXT.
Scc	SET CONDITIONALLY
STOP	STOP
SUB	SUBTRACT
SUBA	SUBTRACT ADDRESS
SUBI	SUBTRACT IMMMEDIATE
SUBQ	SUBTRACT QUICK
SUBX	SUBTRACT WITH EXTEND
SWAP	SWAP REGISTER WORDS
TAS	TEST OPERAND AND SET
TRAP	TRAP
TRAPcc	TRAP CONDITIONALLY
TRAPV	TRAP ON OVERFLOW
TST	TEST OPERAND
UNLK	UNLINK
UNPK	UNPACK BCD

cpBcc	BRANCH CONDITIONALLY
cpDBcc	TST COP COND DECR AND BRA
cpGEN	COPROCESSOR GENERAL INSTR.
cpRESTORE	RESTORE INTERN. STATE OF CP
cpSAVE	SAVE INTERN. STATE OF COPRO
cpScc	SET CONDITIONALLY
cpTRAPcc	TRAP CONDITIONALLY

APPENDIX E: MC68881 REGISTERS AND DATA TYPES

I. MC68881 REGISTERS

The programming model of the MC68881 contains four groups of registers.

A. Floating Point Data Registers (FP0-FP7)

The eight 80-bit floating point data registers are used to store external operands in extended precision format. All external operands are converted to extended precision numbers, regardless of their data format, before they are stored in the floating point data registers. The higher order 16 bits are not used in the extended precision data format.

- The bit field descriptions for extended precision data format;

0 through 51	: Fraction
52 through 62	: Biased Exponent
63	: Sign
64 through 79	: Not used

B. Floating Point Control Register (FPCR)

This 32-bit register is used to enable/disable traps for floating point exceptions and to set rounding mode (Figure 35). The high-order 16 bits are reserved for future use. The low-order 16 bits contain exception enable byte and mode control byte. The user can read from and write to the control register (with high-order word zero for future compatibility).

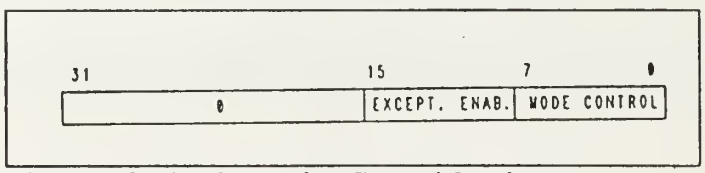


Figure 35 Floating Point Control Register

1. Exception Enable Byte

The exception enable byte contain eight enable bits for each class of floating point exceptions as follows (see Figure 36):

- Exception enable byte bit description :

Bit 15 : BSUN (Branch/Set on Unordered)
Bit 14 : SNAN (Signalling Not A Number)
Bit 13 : OPERR (Operand Error)
Bit 12 : OVFL (Overflow)
Bit 11 : UNFL (Underflow)
Bit 10 : DZ (Divide by Zero)
Bit 9 : INEX2 (Inexact Operation)
Bit 8 : INEX1 (Inexact Decimal Input)

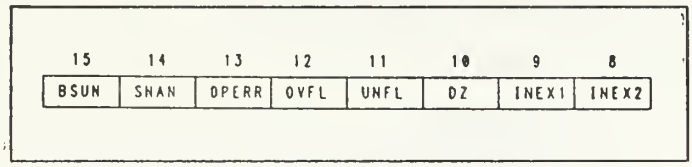


Figure 36 FPCR Exception Enable Byte

The bit numbers in Figure 36 refer to the bit numbers of low-order word of the control register. The status of any bit position determines whether the corresponding exception will be processed or not. To ensure that the exception will be processed, the bit positions for this exception in both the control and status register should be set. The enable byte in the control register should be set before an exception occurs. Setting any enable bit in the control register after an exception occurs does not have any effect in processing the exception, regardless of the corresponding bit value in the status register. The following exceptions can be caused simultaneously by executing a single instruction.

- SNAN and INEX1
- OPERR and INEX2
- OPERR and INEX1
- OVFL and INEX2 and/or INEX1
- UNFL and INEX2 and/or INEX1

In case of multiple exceptions, only the higher priority exception will be processed and the other(s) will be ignored. The bit position of an exception determines its priority, BSUN (Bit 15) has the highest priority.

2. Mode Control Byte

This byte controls the rounding mode and precision. If all the bits are zero then IEEE default is selected.

Bits 7 and 6 determine the rounding precision as follows:

- Bit 7 Bit 6 Precision

0	0	Extended (round to 64 bits)
0	1	Single (round to 24 bits)
1	0	Double (round to 53 bits)
1	1	Undefined

Bits 5 and 4 determine the rounding mode as follows:

- Bit 5 Bit 4 Mode

0	0	To nearest
0	1	Toward zero
1	0	Toward minus infinity
1	1	Toward plus infinity

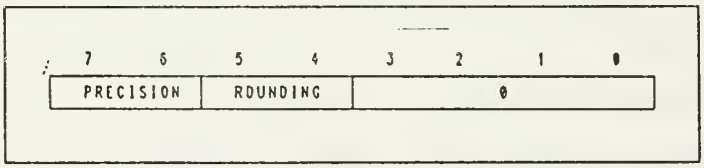


Figure 37 FPCR Mode Control byte

The bit numbers in Figure 37 refer to the bit numbers in the control register. The low order nibble of the mode control byte is always zero.

C. Floating Point Status Register (FPSR)

This 32-bit register contains condition code byte, accrued condition code byte, exception status byte and quotient byte (Figure 38). The user can read from and write to the status register.

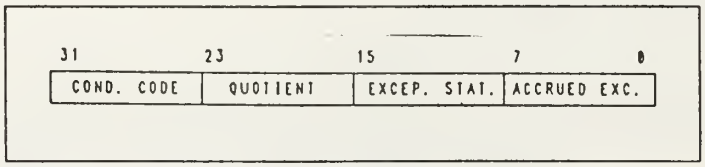


Figure 38 FPCR Status Register

In the following, the bit numbers refer to the bit numbers in the status register.

1. Condition Code Byte

All floating-point arithmetic instructions affect the four bits contained in the status register (see Figure 39). The bits 31 through 28 are reserved and not used. They should be set to zero. The bits 27 through 24 are encoded as follows:

- Condition Code Byte

  Bit 27 : N (Negative)
  Bit 26 : Z (Zero)
  Bit 25 : I (Infinity)
  Bit 24 : NAN (Not A Number or Unordered)

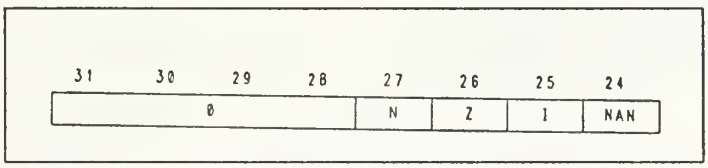


Figure 39 FPSR Condition Code byte

2. Quotient Byte

The sign and the seven least significant bits of the quotient (unsigned) after an FMOD or FREM instruction are stored in the quotient byte (Figure 40).

- Quotient Byte

  Bit 23 : S (Sign)
  Bits 22 through 16 : Q (Quotient)

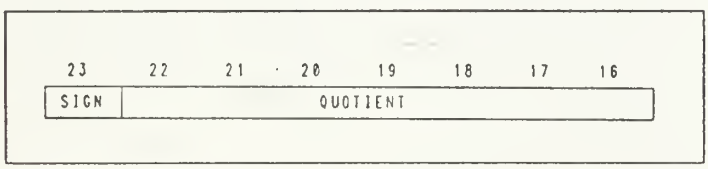


Figure 40 FPSR Quotient byte

The quotient byte remains unaffected until another FMOD or FREM instruction overwrites the byte or it is cleared by the user.

3. Exception Status Byte

Each bit position in the exception status byte indicates the occurence of a floating-point exception, during the last arithmetic or move instruction (Figure 41). This byte is cleared before executing an instruction that can generate a floating point exception, except for FMOVEM and FMOVE control register instructions. Setting a bit in the exception status byte by a user write does not cause an exception.

- Exception status byte bit description:

  Bit 15 : BSUN (Branch/Set on Unordered)
  Bit 14 : SNAN (Signalling Not A Number)
  Bit 13 : OPERR (Operand Error)
  Bit 12 : OVFL (Overflow)
  Bit 11 : UNFL (Underflow)
  Bit 10 : DZ (Divide by Zero)
  Bit 9 : INEX2 (Inexact Operation)
  Bit 8 : INEX1 (Inexact Decimal Input)

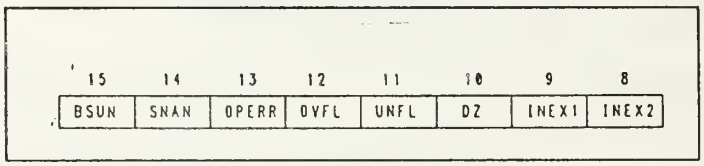


Figure 41 FPSR Exception Status byte

4. Accrued Exception Byte

This byte contains five exception status bits that are logical combinations of the bits in the exception status byte (Figure 42). Unlike the exception status byte, this byte is not cleared before every instruction that can generate an exception. It is cleared either by the user via a write operation to the status register or by the MC68881 via a reset/ a null state size restore operation.

- Accrued exception byte bit description:

  Bit 7 : IOP (Invalid Operation)
  Bit 6 : OVFL (Overflow)
  Bit 5 : UNFL (Underflow)
  Bit 4 : DZ (Divide by Zero)
  Bit 3 : INEX (Inexact)

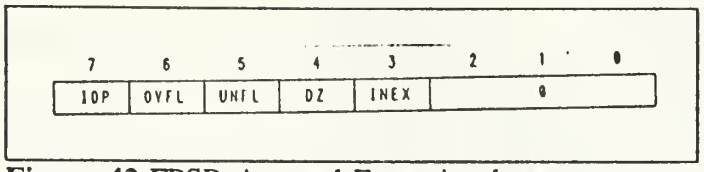


Figure 42 FPSR Accrued Exception byte

Bits 0 through 2 are not used and should be set to zero. The logical combination of the bits are as follows:

A(IOP) = A(IOP) + E(BSUN) + E(SNAN) + E(OPERR)

A(OVFL) = A(OVFL) + E(OVFL)

A(UNFL) = A(UNFL) + (E(UNFL) • E(INEX2))

A(DZ) = A(DZ) + E(DZ)

A(INEX) = A(INEX) + E(INEX1) + E(INEX2) + E(OVFL)

where A( ) = Accrued Exception Byte

E( ) = Exception Status Byte

"+" = Logical OR

"•" = Logical AND.

D. Floating Point Instruction Address Register (FPIAR)

This 32-bit address register is loaded with the address of the floating-point instruction before it is executed. This is due to the non-sequential instruction execution by the MC68020 and MC68881, in which the program counter value saved by the MC68020 in response to a floating-point exception trap may not correspond to the offending instruction. The content of instruction address register can be used by floating-point exception handler to locate the instruction that caused the exception. The instructions that do not modify FPIAR can be used in the exception handler to read the FPIAR without changing the old value. These instructions are FMOVE to/from FPCR, FPSR, FPIAR and FMOVEM. The FPIAR is cleared by a reset or null state size restore operation.

II. MC68881 DATA FORMATS AND TYPES

The MC68881 supports the following data formats;

- Byte Integer (8 bits)

- Word Integer (16 bits)
- Long Word Integer (32 bits)
- Single Precision Real (32 bits)
- Double Precision Real (64 bits)
- Extended Precision Real (96 bits)
- Packed Decimal Real (96 bits)

The integer data formats are straightforward and they are not described in this section. The bit field descriptions for floating data formats are as follows (see Figures 43 through 46):

1. Single Real (32 bits)

- Bit Fields :
  Bit 31 : Sign of Fraction
  Bits 23 through 30 : Exponent
  Bits 0 through 22 : Fraction

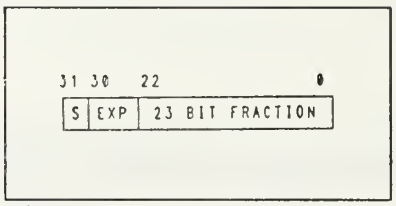


Figure 43 Single Real data format

2. Double Real (64 bits)

- Bit Fields :
  Bit 63 : Sign of Fraction
  Bits 52 through 62 : Exponent
  Bits 0 through 51 : Fraction

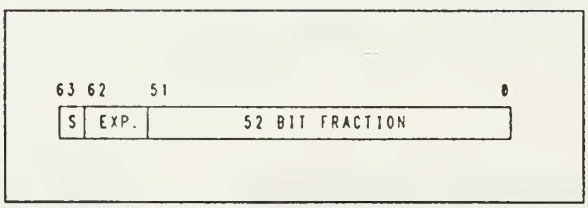


Figure 44 Double Real data format

3. Extended Real (96 bits)

- Bit Fields :
  Bit 95 : Sign of Mantissa
  Bits 81 through 94 : Exponent
  Bits 64 through 80 : Not used (all zeros)
  Bits 0 through 63 : Mantissa

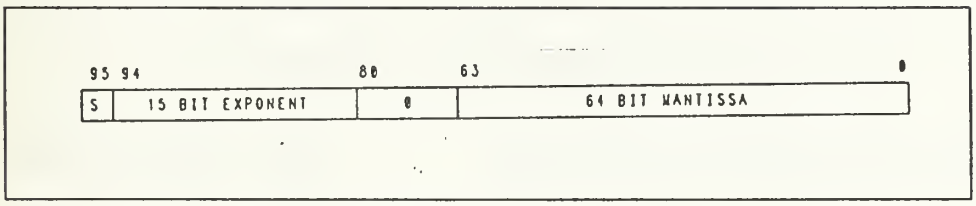


Figure 45 Extended Real data format

4. Packed Decimal Real (96 bits)

- Bit Fields :
  Bit 95 : Sign of Mantissa
  Bit 94 : Sign of Exponent
  Bits 93 through 92 : Used only for infinity and NANs, zero otherwise
  Bits 81 through 91 : Exponent
  Bits 64 through 80 : Zero (if no overflow in BIN to DEC conversion)
  Bits 0 through 63 : Mantissa

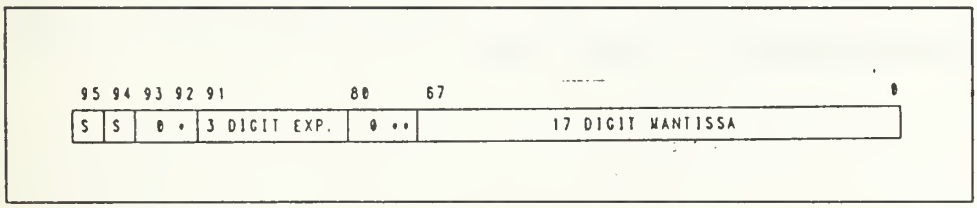


Figure 46 Packed Decimal Real data format

The single, double and extended precision floating-point data formats can represent five floating-point data types which have three parts: Sign of mantissa, Exponent and Mantissa.

- Normalized Numbers (Figure 47)

  Sign of Mantissa : 0 or 1
  Exponent : Greater Than MINIMUM, Less Than MAXIMUM
  Mantissa : Any bit pattern

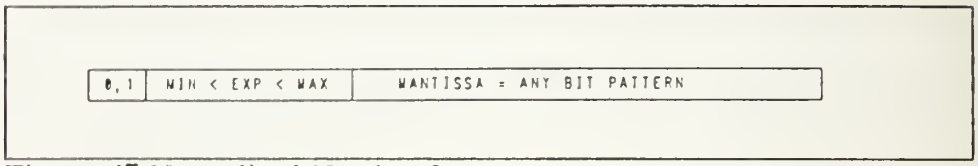


Figure 47 Normalized Number format

- Denormalized Numbers (Figure 48)

  Sign of Mantissa : 0 or 1
  Exponent : 0
  Mantissa : Any non-zero bit pattern

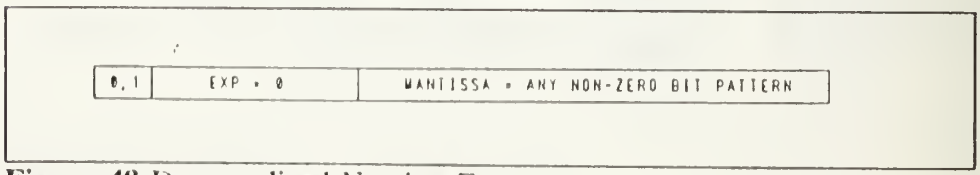


Figure 48 Denormalized Number Format

- Zeros (Figure 49)

  Sign of Mantissa : 0 or 1
  Exponent : 0
  Mantissa : 0

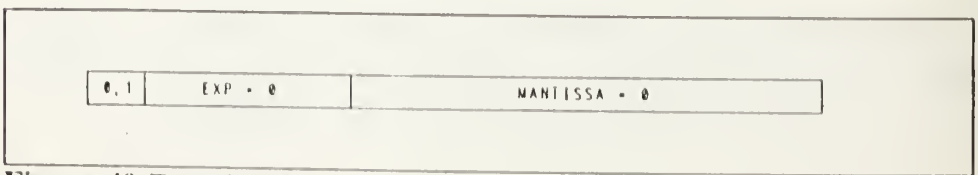


Figure 49 Zero format

- Infinities (Figure 50)

  Sign of Mantissa : 0 or 1
  Exponent : MAXIMUM
  Mantissa : 0

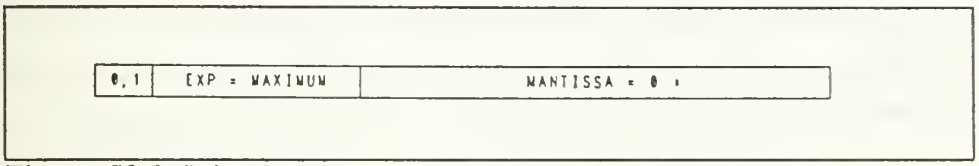


Figure 50 Infinity format

- Not-A-Number (Figure 51)

  Sign of Mantissa : 0 or 1
  Exponent : MAXIMUM
  Mantissa : 0

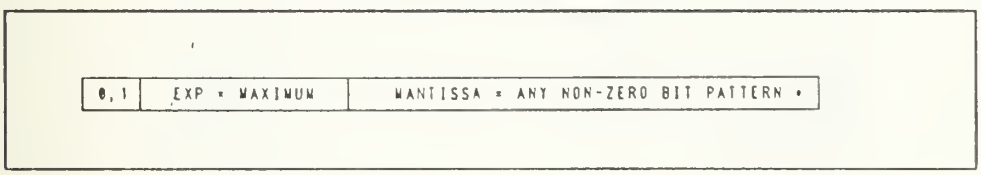


Figure 51 Not-A-Number format

APPENDIX F: MC68881 COPROCESSOR INTERFACE

A. SIGNAL CONNECTION AND COPROCESSOR ACCESS

The MC68881 is connected to the main processor via 32-bit data bus, as shown in the Figure 52. The pins A0 and SIZE are both pulled-up to Vcc in order to configure 32-bit data bus connection. All the other signals, except for the chip select, are directly connected to the corresponding pins of the main processor. The chip select signal (!CS) is generated from A18, A17 and A15 by the external logic given in **Appendix G**.

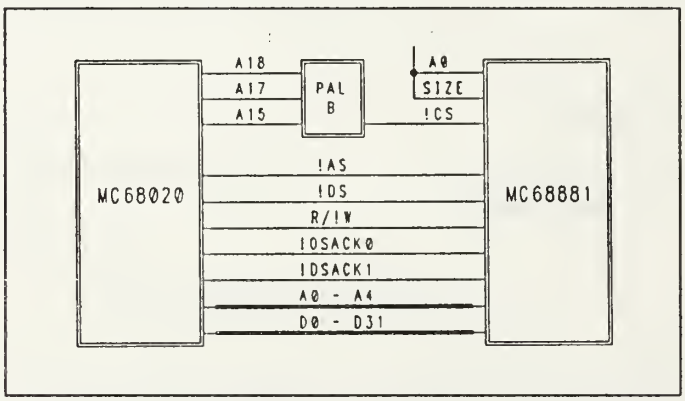


Figure 52 MC68020/MC68881 32 bit data bus connection

For coprocessor access, the address lines A0 through A4 and A13 through A19 are encoded as follows (see Figure 53):

- A0 through A4 : Indicate the Coprocessor Interface Register to be accessed
- A13 through A15 : Indicate the ID number of the coprocessor to be accessed
- A16 through A19 : Indicate that CPU space transaction is coprocessor communications. (0010)

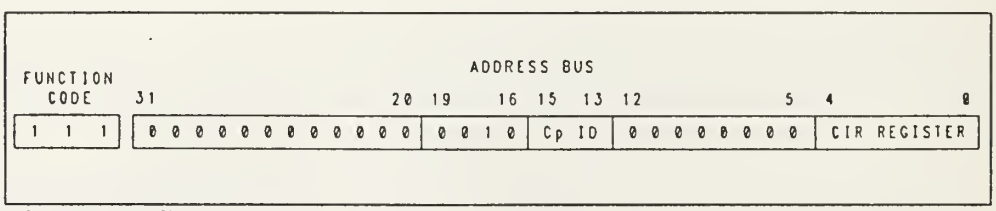


Figure 53 CPU space encoding for coprocessor access.

B. COPROCESSOR INTERFACE REGISTERS

The main processor communicates with the MC68881 via a group of coprocessor interface registers which are either 16-bit or 32-bit long. The 16-bit interface registers are placed on the high order word of 32-bit data bus (D31-D16) by asserting !DSACK1 and negating !DSACK0, regardless of the value of A1. Figure 54 gives a list of coprocessor interface registers with their address offsets, widths and read/write attributes. Write access to a read-only register is ignored, whereas read access to a write-only register returns all ones. The registers Operation Word (Offset 08 Hex) and Operand Address (Offset 1C Hex) are not used by the MC68881.

	A4	A3	A2	A1	A0	OFFSET	WIDTH	TYPE
RESPONSE	0	0	0	0	X	$00	16	READ
CONTROL	0	0	0	1	X	$02	16	WRITE
SAVE	0	0	1	0	X	$04	16	READ
RESTORE	0	0	1	1	X	$06	16	R/W
OPERATION WORD	0	1	0	0	X	$08	16	R/W
COMMAND	0	1	0	1	X	$0A	16	WRITE
(RESERVED)	0	1	1	0	X	$0C	16	-
CONDITION	0	1	1	1	X	$0E	16	WRITE
OPERAND	1	0	0	X	X	$10	32	R/W
REGISTER SELECT	1	0	1	0	X	$14	16	READ
(RESERVED)	1	0	1	1	X	$16	16	-
INSTRUCTION ADDRESS	1	1	0	X	X	$18	32	WRITE
OPERAND ADDRESS	1	1	1	X	X	$1C	32	R/W

Figure 54 Coprocessor Interface Register map

a. Response CIR ($00)

The response CIR is used to transfer service requests from the MC68881 to the main processor. The MC68881 does not start instruction execution until the main processor reads the Response CIR for the first time after a write to the Command CIR.

b. Control CIR ($02)

The control CIR is used by the main processor to issue an instruction abort or an exception acknowledge to the MC68881. The high order 14 bits of the Control CIR are not used. Although bits 0 and 1 are defined as **abort** and **exception acknowledge**, respectively, it has the same effect on the MC68881 to set bit 0 or bit 1. After a write to the Control CIR, the MC68881 takes the following steps;

- Terminates the instruction execution.
- Clears pending exceptions, if any.
- Resets the bus interface and gets ready to begin new instruction protocol.

c. Save CIR ($04)

The main processor uses the Save CIR to issue a context save command to the MC68881 and to read the format word of the MC68881 state frame. A read from this register suspends the operation currently being executed by the MC68881 and initiates a state save operation. If the current operation is a state save or state restore, then it will not be suspended by a read from the Save CIR.

d. Restore CIR ($06)

The Restore CIR is used by the main processor to transfer a context restore command to the MC68881 and to validate the format word of a state frame. After a write to this register, the MC68881 stops executing any operation and prepares to load new internal state context from memory.

e. Operation Word CIR ($08)

This register is not used by the MC68881. A write to this register is ignored and it does not cause a protocol violation.

f. Command CIR ($0A)

The communication for executing a general coprocessor instruction (cpGEN) is initiated by a write to the Command CIR by the main processor. When a write to this register is detected, the MC68881 latches the data from the data bus, and, if not busy executing a previous instruction, the response CIR is encoded with the first primitive of the dialog for the execution of the new instruction. Otherwise, the latched data is saved for future use and the response CIR is encoded with the null primitive.

g. Condition CIR ($0E)

The use of this register is the same as the Command CIR, except that the Condition CIR is for conditional coprocessor instructions. The value of the conditional evaluation is returned to the main processor with the first primitive of the dialog.

h. Operand CIR ($10)

The 32-bit Operand CIR is used to transfer data between the main processor and the MC68881. An access to the Operand register by MC68881 is legal after reading the following primitives:

- Evaluate effective address and transfer data
- Transfer multiple coprocessor registers
- Transfer single main processor register

and after a read/write of idle or busy format word from/to the save/restore CIR. An access to this register in other cases causes a protocol violation.

i. Register Select CIR ($14)

The Register Select CIR is read by the main processor to get the register mask during a move multiple floating-point data register operation. An access to this register is legal only just after issuing a transfer multiple coprocessor registers primitive to the main processor. An access at any other time causes a protocol violation. Only low-order eight bits of this register are used.

j. Instruction Address CIR ($18)

The main processor uses this 32-bit register to transfer the address of the MC68881 instruction already being executed when the PC bit of any primitive is set. An access to the Instruction Address CIR at any time does not cause a protocol violation. FPIAR register is updated, whenever a write to the Instruction Address CIR occurs. A read from this register returns all ones.

k. Operand Address CIR ($1C)

This register is not used by the MC68881 and an access to this register does not cause a protocol violation. Reads from this register always return all ones and writes are ignored.

C. COPROCESSOR COMMUNICATION AND RESPONSE PRIMITIVES

1. Coprocessor Communication

The length of MC68881 instructions vary between one to eight words. The first two words are called operation word and coprocessor command word. The words after the coprocesor command word specify the operands. Bits 12 through 15 in the operation word are always one, which specify F-line operation code. Bits 9 through 11 indicate the coprocessor ID. The low order byte of the operation word is encoded according to the type of the instruction.

The MC68020 and MC68881 follow the communication protocol, given below, during the execution of a floating-point instruction:

- The MC68020 detects an F-line operation word and initiates the communication by writing to the appropriate coprocessor interface register (or by a read for MC68881 save instruction).
- The MC68881 gives a response to the previous write operation by writing, what is called a primitive, to the response CIR. The MC68020 then reads the response CIR and proceeds in accordance with one of the following indications by the response primitive:

The MC68881 is busy:	Process any pending interrupt, query the MC68881 again.

There is an exception condition and MC68020 is instructed to take an exception:

Acknowledge the exception and initiate the processing.

The MC68881 requests service:

Perform the service requested by the MC68881 such as;
Evaluate the effective address.
Transfer data between effective address and the MC68881.
Query MC68881 after performing the service.

The execution of the coprocessor instruction can start and MC68020 is released:

Begin the execution of the next instruction.
If in trace mode, take the trace exception after coprocessor instruction is processed.

2. Response Primitives.

The response primitive is the data read from the coprocessor interface response CIR. There are 18 response primitives defined by the MC68000 family coprocessor interface. The MC68881 uses six of these primitives. The response primitives are 16-bit words and have the following general format;

- Bit 15 (CA): Come Again; if set, the MC68020 should return to read the response CIR again, after performing the service requested by MC68881.
- Bit 14 (PC): Program Counter; if set, the MC68020 should immediately pass the current PC value to the instruction address CIR.
- Bit 13 (DR): Direction; if set, it indicates a main processor read, otherwise indicates a main processor write.
- Bits 0 through 12: Contains data dependent on the individual primitive.

The following are the six primitives used by the MC68881:

a. Null Primitive

The null primitive provides synchronization and concurrent execution with the main processor. Only five bits are used to encode the null response, the remaining bits, except for bit 11, are all zeros (Figure 55):

- Bit 15 (CA) : Come Again; as explained above.
- Bit 14 (PC) : Program Counter; as explained above.
- Bit 8 (IA) : Interrupt Acknowledge; when set, the main processor may process any pending interrupt, otherwise interrupts are ignored.
- Bit 1 (PF) : Indicates the status of the MC68881; when set, the MC68881 is idle. It is cleared if the MC68881 is executing an instruction.
- Bit 0 (TF) : Indicates the result of a conditional evaluation.

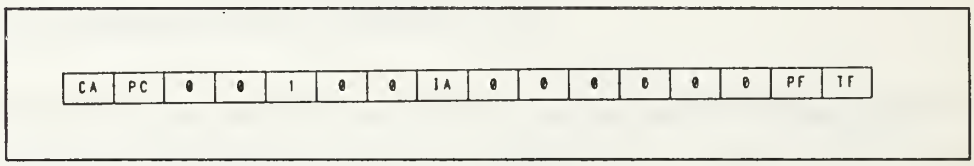


Figure 55 Null Format

b. Evaluate Effective Address and Transfer Data

The MC68881 uses this primitive to request the transfer of data between its data or control registers and an external location, which can be either a memory location or a register of the main processor. The bits 13 through 15 are DR, PC and CA bits as explained in the general format. The bit 12 is set to one, and bit 11 to zero (Figure 56).

- Bits 8 through 10 specifies one of the following addressing modes:

000 : Control Alterable
001 : Data Alterable
010 : Memory Alterable
011 : Alterable
100 : Control
101 : Data
110 : Memory
111 : Any Effective Address

If the class of effective address in the operation word does not match the specified class, then the main processor should write an abort command to the control CIR.

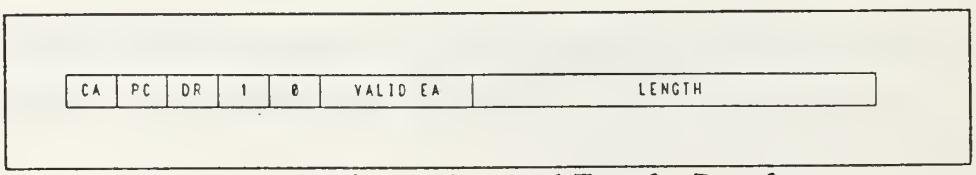


Figure 56 Evaluate Effective Address and Transfer Data format

c. Transfer Single Main Processor Register

The MC68881 requests the transfer of one main processor register by using this primitive. The MC68020 writes a long word to the operand CIR in response to this primitve. The CA, PC and DR bits have the same functions as explained above. Bits 0 through 2 indicate the register number to be transferred, and bit 3 (D/A) specifies whether it is a data (D/A=0) or address (D/A=1) register. Bits 10 and 11 are set to one; all the other bits are zeros (Figure 57).

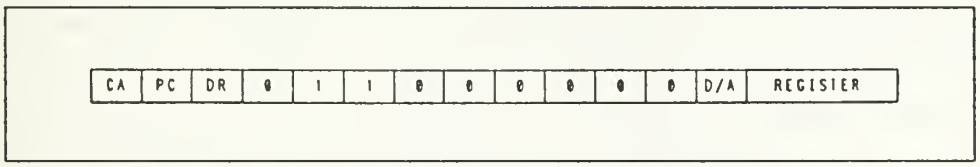


Figure 57 Transfer Single Main Processor Register format

d. Transfer Multiple Coprocessor Register

The MC68881 uses this primitive to request the transfer of multiple floating-point registers to or from memory. Bits 13 through 15 are DR, PC and CA bits. Bits 0 through 7 indicate the size, in bytes, of the registers to be transferred. The MC68881 registers are always 12 bytes long. Bit 8 is set to one and all the other bits are zeros (Figure 58).

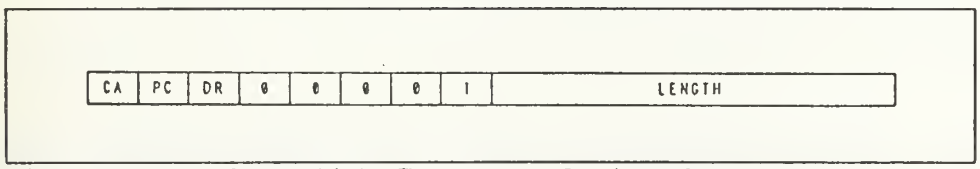


Figure 58 Transfer Multiple Coprocessor Register format

e. Take Pre-Instruction Exception

This primitive is used in the following cases:

- When an arithmetic or conditional instruction is initiated, and there is a pending exception from a previously executed concurrent instruction.
- When an illegal command word is written to the command CIR, or a protocol violation occurs.
- When a conditional instruction which utilizes one of the IEEE non-aware conditional predicates is executed, and the NAN bit in FPSR is one.

The CA and DR bits are zero. The PC bit is zero, when the execution of a new instruction is preemptied by the exception. The PC bit is one, when the exception is generated by an illegal command word or when the exception is reported during a conditional instruction execution. The bits 0 through 7 indicates the type of the exception which is used by the main processor to calculate the address of the exception handler. The bits 8 and 9 are zero, and all the other bits are set to one (Figure 59).

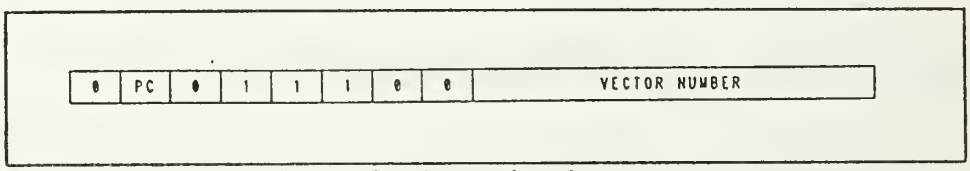


Figure 59 Take Pre-instruction Exception format

f. Take Mid-Instruction Exception

The MC68881 uses this primitive, if an exception occurs during the execution of FMOVE FPm, <ea> instruction. In the format of this primitive, the CA, PC and DR bits are set to zero. Bits 0 through 7 contain the vector number which identifies the type of exception. Bit 9 is zero, and all the other bits are ones (Figure 60).

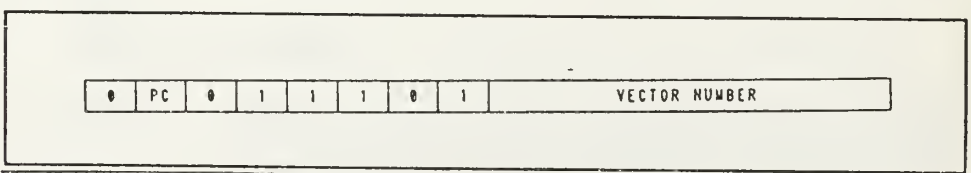


Figure 60 Take Mid-instruction Exception format

APPENDIX G: DESIGN OF THE ECB

A. Memory Mapping

The memory is divided into three segments:

- First segment : $00000 - $1FFFF
- Second segment : $20000 - $3FFFF
- Third segment : $40000 - $7FFFF

There are two memory mapping schemes which differ from each other in how the segments are accessed. The first scheme is defaulted after reset or power-up. The only way to switch from the first scheme to the second is to make a coprocessor access. An external reset should be applied in order to switch back to the first scheme.

The memory map in scheme 1 is shown in Table 16.

Table 16 ECB Memory Mapping Scheme 1.

	ADDRESS RANGE	READ	WRITE
SEGMENT 1	$00000 - $1FFFFF	ROM	RAM
SEGMENT 2	$20000 - $3FFFFF	COP	COP
SEGMENT 3	$40000 - $7FFFFF	ROM	

In scheme 1, both ROM and RAM are mapped to Segment 1. RAM is accessed for writing only and all reads are from ROM. Segment 2 can be accessed for both writing and reading. ROM can also be accessed in the higher addresses. The primary area for ROM is Segment 3. The ROM in the low addresses can be thought as an image of the ROM in the high addresses. This image is created and removed by the signal, called

PHANTOM. Mapping the ROM to Segment 1 allows to access the initialization routines after reset or power-up.

The memory map in scheme 2 is given in Table 17.

Table 17 ECB Memory Mapping Scheme 2

	ADDRESS RANGE	
SEGMENT 1	$00000 - $1FFFFF	RAM
SEGMENT 2	$20000 - $3FFFFF	CDP
SEGMENT 3	$40000 - $7FFFFF	ROM

In the second scheme, the image of ROM is removed from Segment 1. RAM can be accessed for both reading and writing. This is the condition in normal operation of the ECB.

B. Programmable Array Logic circuit PAL B

Figure 61 shows how the chip select and other control signals are generated for the memory mapping schemes. All the signals are the outputs of the PAL B (PAL16L8). This PAL has been programmed by ABEL software. Appendix H includes the programming files of the PAL B.

	A18	A17	A18	R/$\overline{W}$	PHAN	A1	A1	S1	S1	$\overline{DS}$
CopE	X	X	X	X	X	1	1	X	X	1
ROMCE	X	X	X	X	X	X	X	X	X	X
	X	0	X	1	0	X	X	X	1	0
RAMCE	X	X	X	0	1	X	1	1	1	X
	0	0	X	X	0	X	X	X	X	0
RAMOE	0	0	X	1	0	X	X	X	X	X
RAM1W	0	0	X	0	X	0	0	X	X	0
RAM2W	0	0	X	0	X	0	1	X	X	0
	0	0	X	0	X	0	X	X	0	0
	0	0	X	0	0	0	0	0	0	0
RAM3W	0	0	X	0	X	1	0	X	0	0
	0	0	X	0	X	0	1	X	0	0
	0	0	X	0	X	0	X	0	X	0
	0	0	X	0	X	0	X	1	1	0
RAM4W	0	0	X	0	X	1	1	X	X	0
	0	0	X	0	X	1	X	1	X	0
	0	0	X	0	X	X	X	0	0	0
	0	0	X	0	X	X	1	1	1	0

Figure 61 Generation of the memory mapping signals

C. Programmable Array Logic Circuit PAL A

The PAL A generates the signals required for interfacing the MC68020 with memory and RS-232 port.

1. The PHANTOM Signal

The **PHANTOM** signal is used to create and remove an image of the ROM in Segment 1. During power-up or reset, asserting the **!RESET** line sets the **PHANTOM** output high. This output remains high after the **!RESET** input is negated, until a

coprocessor access occurs, i.e., the **!CopE** input is asserted. It is the responsibility of initalization routine to assert the **!CopE** input by making an access to Segment 2. (see Reference 1). The **!CopE** input is synchronized with the **!AS** signal.

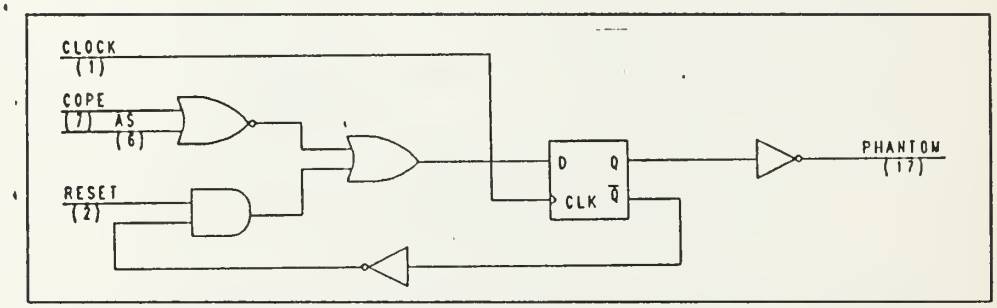


Figure 62 PHANTOM signal generation.

2. RS232 Transmit/Receive Circuit

The **!INTERRUPT** output, which is connected to the **!IPL2** input of the MC68020, indicates that data is being received on RS-232 line. The **!INTERRUPT** output is not asserted, unless the address lines **A19** and **A17** are set high, even if there is an incoming data on RS-232 line. It is the responsibility of the communication routine to monitor the RS-232 line by setting the address lines **A19** and **A17**. (see Reference 1).

The **!RS232OUT** output is used to transmit data on RS-232 line, by asserting and negating the address lines **A19** and **A15** under software control. (See Reference 1.)

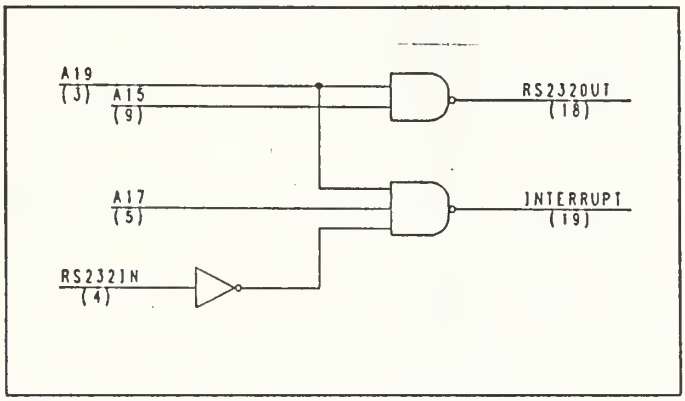


Figure 63 RS232 Transmit/Receive Circuit

3. Data Size And Transfer Acknowledge Signals

The DSACK signals return 8-bit port size for the ROM, and 32-bit port size for the RAM. Any access to the coprocessor does not cause the DSACK signals to be asserted, as the MC68881 provides its own port size. If the ROM is accessed, only the DSACK0 output is asserted to return an 8-bit port size. The outputs W0, W1 and W2 are used to provide a delay of eight clock cycles, before asserting the DSACK0, when the ROM is accessed. This is because the ROM chip has a longer delay (150 ns for AMD 27C256 chip) than the RAM chips (55 ns for Motorola 6164 chip). Both DSACK signals are asserted, without any forced delay, when the RAM is accessed.

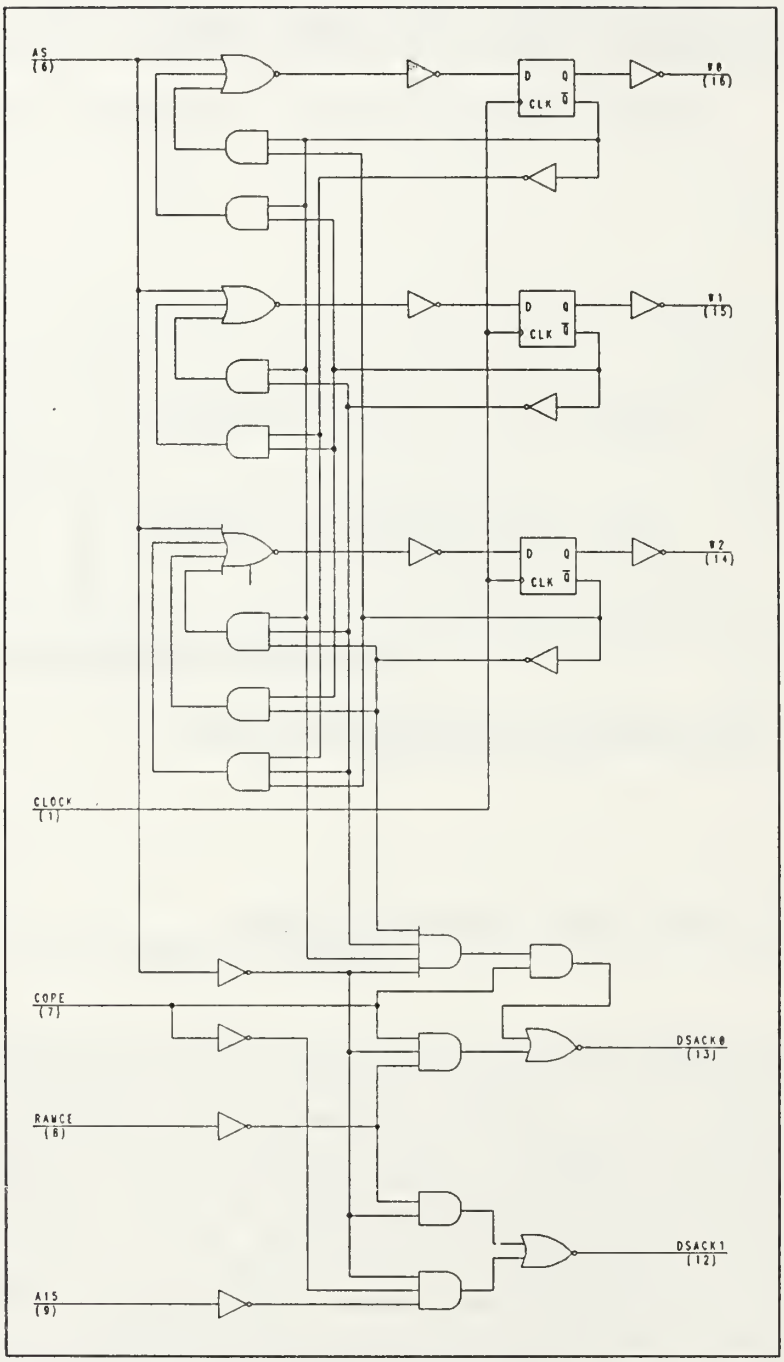


Figure 64 DSACK Signal Generation.

D. Reset Circuit

The reset circuit was built around the Motorola's undervoltage-sensing IC, the MC34064. The output of the circuit is driven low for more than 100 ms, during power-up or when the reset button is pressed, and provides an external reset signal for both the MC68020 and MC68881.

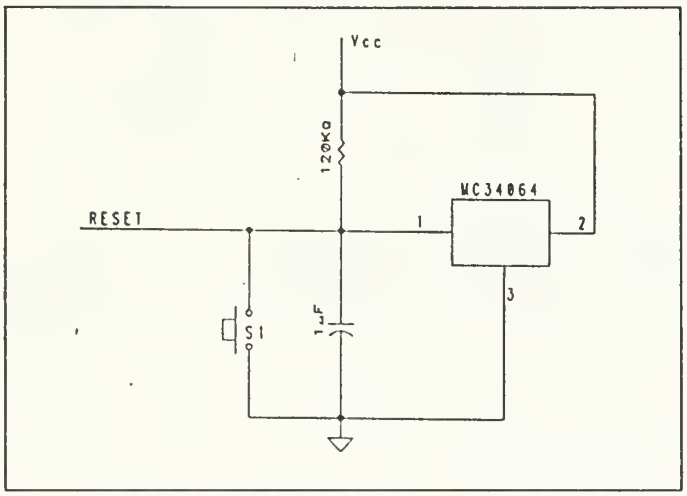


Figure 65 Reset Circuit

E. Software Abort Circuit

The circuit for software abort consists of all passive components, as shown in the following figure. When the switch S1 is pressed, IPL2, IPL1 and AVEC lines are held low for a period of approximately 5 microsecond, which generates an autovectored level 6 interrupt.

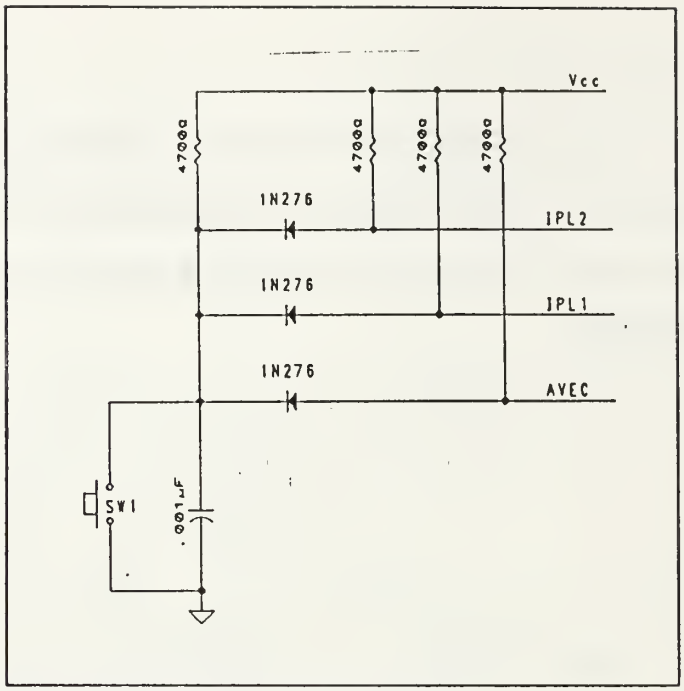


Figure 66 Software Abort circuit

F. I/O Interface for External Devices

All the pads and holes have been provided to install TTL series line drivers 74245 (bidirectional for 8-bit data) and 74244/74241 (unidirectional for address and control lines). The connections for external I/O interface are given in Figure 67. This interface has not been implemented and tested in this thesis. It is left as a future improvement.

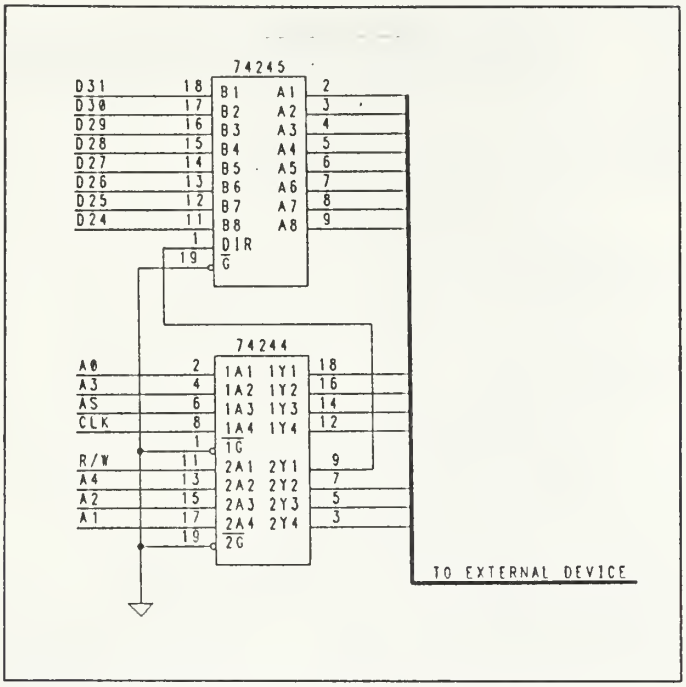


Figure 67 I/O interface for external devices

The complete circuit diagram and two layer PCB layout are given in figures 68 and 69, respectively. The I/O interface for external devices are not included in the circuit diagram.

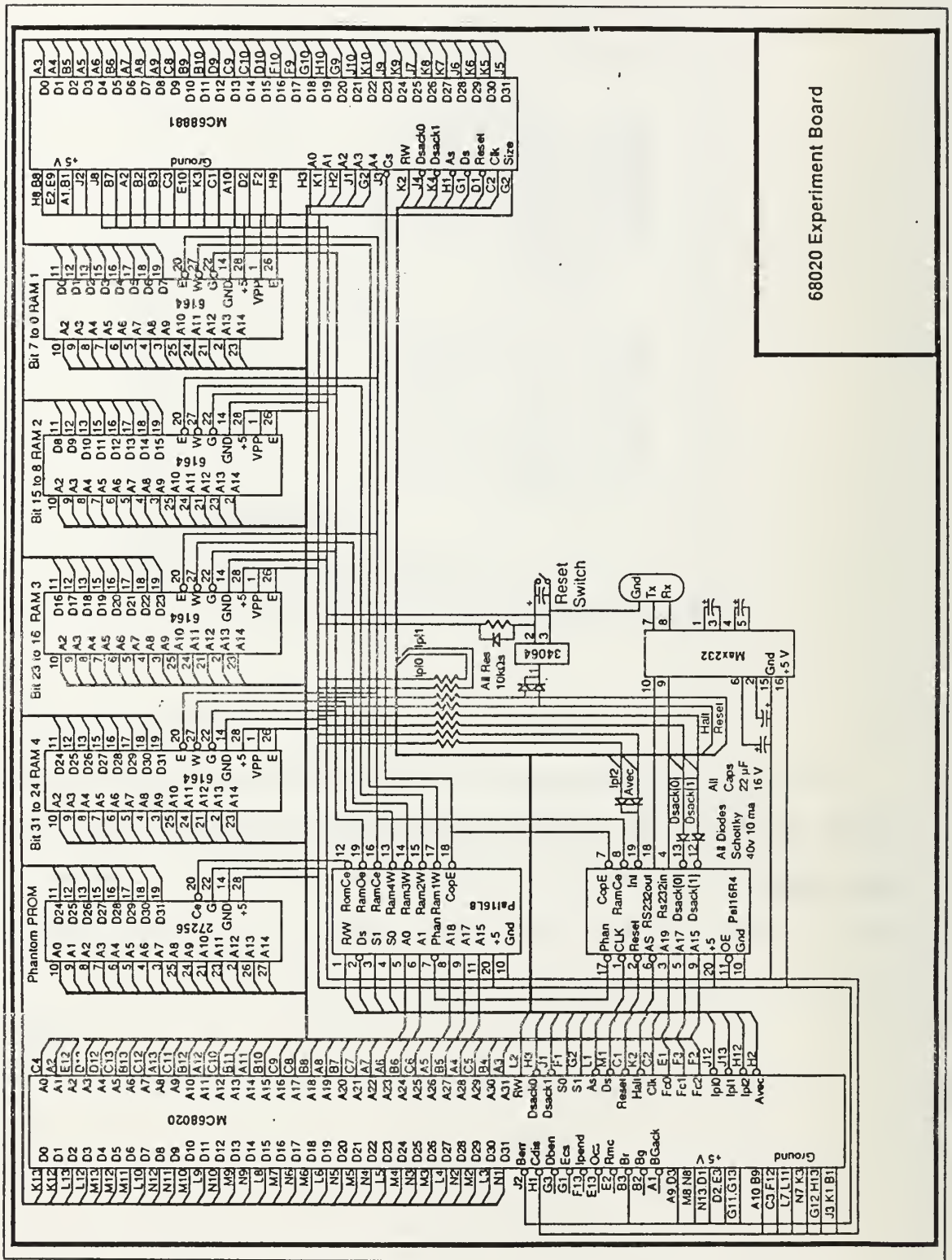


Figure 68 ECB Circuit Diagram.

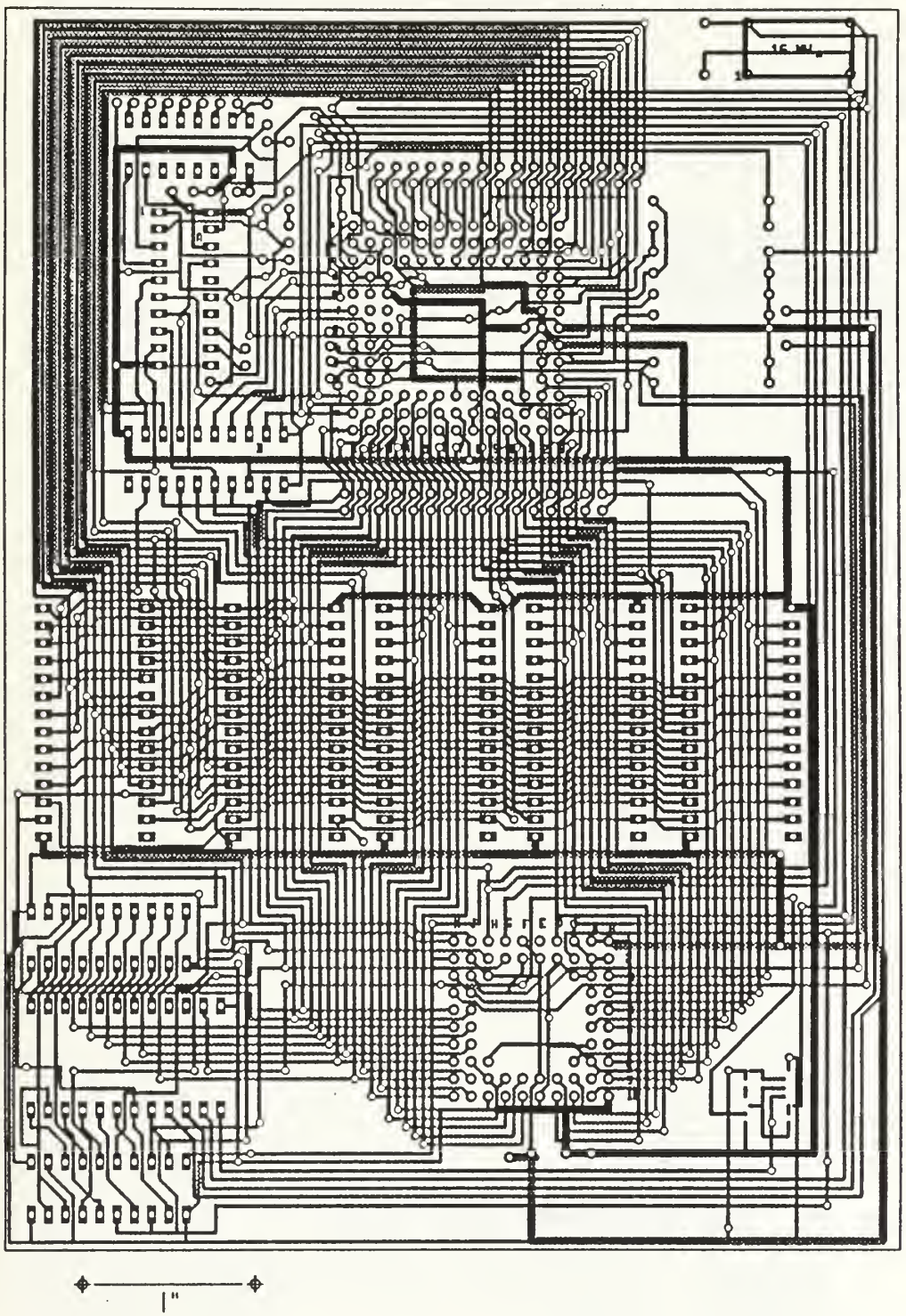


Figure 69 ECB Two Layer PCB Layout

APPENDIX H: PAL A PROGRAMMING FILES

A. PAL A LISTING FILE

```
0001 !module  pala1  ;
0002 !flag '-f' ;
0003 !        PAL_A DEVICE  'P16R4';
0004 !        CLK,COPE,RESET,A19,A17,A15,AS,RSIN   PIN 1,7,2,3,5,9,6,4          ;
0005 !        RAMCE,OE,PHAN,RSOUT,INT              PIN 8,11,17,18,19            ;
0006 !        W0,W1,W2,DSACK0,DSACK1               PIN 16,15,14,13,12           ;
0007 !        CK,X,Z  =  .C.,.X.,.Z.                                            ;
0008 !
0009 !        S0 = ^B000 ;
0010 !        S1 = ^B001 ;
0011 !        S2 = ^B010 ;
0012 !        S3 = ^B011 ;
0013 !        S4 = ^B100 ;
0014 !        S5 = ^B101 ;
0015 !        S6 = ^B110 ;
0016 !        S7 = ^B111 ;
0017 !
0018 !
0019 !TEST_VECTORS (  [CLK,OE,RAMCE,AS,COPE] -> [W0,W1,W2,DSACK0])
0020 !                [CK,0,1,1,0] -> [0,0,0,1] ;
0021 !                [CK,0,1,0,0] -> [1,1,1,1] ;
0022 !                [CK,0,1,0,0] -> [0,1,1,1] ;
0023 !                [CK,0,1,0,0] -> [1,0,1,1] ;
0024 !                [CK,0,1,0,0] -> [0,0,1,1] ;
0025 !                [CK,0,1,0,0] -> [1,1,0,1] ;
0026 !                [CK,0,1,0,0] -> [0,1,0,1] ;
0027 !                [CK,0,1,0,0] -> [1,0,0,1] ;
0028 !                [CK,0,1,0,0] -> [1,0,0,1] ;
0029 !                [CK,0,1,0,0] -> [1,0,0,1] ;
0030 !                [CK,0,0,0,0] -> [1,0,0,1] ;
0031 !                [CK,0,1,1,0] -> [0,0,0,1] ;
0032 !                [CK,1,1,1,0] -> [Z,Z,Z,1] ;
0033 !
0034 !                [CK,0,1,1,1] -> [0,0,0,1] ;
0035 !                [CK,0,1,0,1] -> [1,1,1,1] ;
0036 !                [CK,0,1,0,1] -> [0,1,1,1] ;
0037 !                [CK,0,1,0,1] -> [1,0,1,1] ;
0038 !                [CK,0,1,0,1] -> [0,0,1,1] ;
0039 !                [CK,0,1,0,1] -> [1,1,0,1] ;
0040 !                [CK,0,1,0,1] -> [0,1,0,1] ;
0041 !                [CK,0,1,0,1] -> [1,0,0,0] ;
0042 !                [CK,0,1,0,1] -> [1,0,0,0] ;
0043 !                [CK,0,1,0,1] -> [1,0,0,0] ;
0044 !                [CK,0,0,0,1] -> [1,0,0,0] ;
0045 !                [CK,0,1,1,1] -> [0,0,0,1] ;
0046 !                [CK,1,1,1,1] -> [Z,Z,Z,1] ;
0047 !
0048 !
0049 !TEST_VECTORS (  [CLK,OE,AS,COPE,RESET,PHAN] ->  [PHAN] )
0050 !                [CK,0,0,1,0,0]      ->  [1] ;
0051 !                [CK,0,0,1,0,1]      ->  [1] ;
0052 !                [CK,0,0,1,0,1]      ->  [1] ;
0053 !                [CK,0,0,1,1,1]      ->  [1] ;
0054 !                [CK,0,0,1,1,1]      ->  [1] ;
0055 !                [CK,0,0,1,1,1]      ->  [1] ;
0056 !                [CK,0,0,1,1,1]      ->  [1] ;
0057 !                [CK,0,0,1,1,1]      ->  [1] ;
0058 !                [CK,0,0,1,1,1]      ->  [1] ;
0059 !                [CK,0,0,1,1,1]      ->  [1] ;
```

```
0060 |              [CK,0,0,1,1,1]      ->  [1] ;
0061 |              [CK,0,0,1,1,1]      ->  [1] ;
0062 |              [CK,0,0,1,0,1]      ->  [1] ;
0063 |              [CK,0,0,1,0,1]      ->  [1] ;
0064 |              [CK,0,0,1,0,1]      ->  [1] ;
0065 |              [CK,0,0,0,0,1]      ->  [0] ;
0066 |              [CK,0,0,1,0,0]      ->  [1] ;
0067 |              [CK,0,0,1,1,1]      ->  [1] ;
0068 |              [CK,0,0,1,1,1]      ->  [1] ;
0069 |              [CK,0,0,0,1,1]      ->  [0] ;
0070 |              [CK,0,0,0,1,0]      ->  [0] ;
0071 |              [CK,0,0,1,1,0]      ->  [0] ;
0072 |              [CK,0,0,1,1,0]      ->  [0] ;
0073 |              [CK,0,0,1,0,0]      ->  [1] ;
0074 |              [CK,0,0,1,0,1]      ->  [1] ;
0075 |
0076 |              [CK,0,1,1,0,0]      ->  [1] ;
0077 |              [CK,0,1,1,0,1]      ->  [1] ;
0078 |              [CK,0,1,1,0,1]      ->  [1] ;
0079 |              [CK,0,1,1,1,1]      ->  [1] ;
0080 |              [CK,0,1,1,1,1]      ->  [1] ;
0081 |              [CK,0,1,1,1,1]      ->  [1] ;
0082 |              [CK,0,1,1,1,1]      ->  [1] ;
0083 |              [CK,0,1,1,1,1]      ->  [1] ;
0084 |              [CK,0,1,1,1,1]      ->  [1] ;
0085 |              [CK,0,1,1,1,1]      ->  [1] ;
0086 |              [CK,0,1,1,1,1]      ->  [1] ;
0087 |              [CK,0,1,1,1,1]      ->  [1] ;
0088 |              [CK,0,1,1,0,1]      ->  [1] ;
0089 |              [CK,0,1,1,0,1]      ->  [1] ;
0090 |              [CK,0,1,1,0,1]      ->  [1] ;
0091 |              [CK,0,1,0,0,1]      ->  [1] ;
0092 |              [CK,0,1,1,0,0]      ->  [1] ;
0093 |              [CK,0,1,1,1,1]      ->  [1] ;
0094 |              [CK,0,1,1,1,1]      ->  [1] ;
0095 |              [CK,0,1,0,1,1]      ->  [1] ;
0096 |
0097 |              [CK,0,1,0,1,1]      ->  [1] ;
0098 |              [CK,0,0,0,1,0]      ->  [0] ;
0099 |              [CK,0,1,0,1,0]      ->  [0] ;
0100 |              [CK,0,1,1,1,0]      ->  [0] ;
0101 |              [CK,0,1,1,0,0]      ->  [1] ;
0102 |
0103 |              [CK,0,1,1,0,1]      ->  [1] ;
0104 |
0105 |TEST_VECTORS (  [A19 , A15] ->  [RSOUT] )
0106 |              [0,0]       ->  [1] ;
0107 |              [0,1]       ->  [1] ;
0108 |              [1,0]       ->  [1] ;
0109 |              [1,1]       ->  [0] ;
0110 |
0111 |TEST_VECTORS (  [A19 , A17, RSIN] ->  [INT] )
0112 |              [0,0,0]     ->  [1] ;
0113 |              [0,0,1]     ->  [1] ;
0114 |              [0,1,0]     ->  [1] ;
0115 |              [0,1,1]     ->  [1] ;
0116 |              [1,0,0]     ->  [1] ;
0117 |              [1,0,1]     ->  [1] ;
0118 |              [1,1,0]     ->  [0] ;
0119 |              [1,1,1]     ->  [1] ;
0120 |
0121 |TEST_VECTORS (  [AS, W0,W1,W2,RAMCE,COPE] ->  [DSACK0] )
0122 |              [0,0,0,0,0,1]     ->  [0] ;
0123 |              [0,1,0,0,X,1]     ->  [0] ;
0124 |              [1,X,X,X,X,1]     ->  [1] ;
0125 |              [1,X,X,X,X,1]     ->  [1] ;
0126 |              [X,X,X,X,X,0]     ->  [1] ;
0127 |
0128 |
```

```
0129 !TEST_VECTORS (  [AS, A15, COPE, RAMCE] ->  [DSACK1] )
0130 !                [0,0,0,0]     ->  [0] ;
0131 !                [0,0,0,1]     ->  [1] ;
0132 !                [0,0,1,0]     ->  [0] ;
0133 !                [0,0,1,1]     ->  [1] ;
0134 !                [0,1,0,0]     ->  [0] ;
0135 !                [0,1,0,1]     ->  [0] ;
0136 !                [0,1,1,0]     ->  [0] ;
0137 !                [0,1,1,1]     ->  [1] ;
0138 !                [1,0,0,0]     ->  [1] ;
0139 !                [1,0,0,1]     ->  [1] ;
0140 !                [1,0,1,0]     ->  [1] ;
0141 !                [1,0,1,1]     ->  [1] ;
0142 !                [1,1,0,0]     ->  [1] ;
0143 !                [1,1,0,1]     ->  [1] ;
0144 !                [1,1,1,0]     ->  [1] ;
0145 !                [1,1,1,1]     ->  [1] ;
0146 !
0147 !
0148 !EQUATIONS
0149 !        RSOUT   = ! (A19 & A15)                                  ;
0150 !        INT     = ! (A19 & A17 & !RSIN)                          ;
0151 !        DSACK0  = AS # ( RAMCE & (! W0 # W1 # W2 ) ) # !COPE     ;
0152 !        !DSACK1 = ( !AS & !RAMCE ) # ( !AS & !COPE & A15 )       ;
0153 !        !PHAN  := (!(COPE # AS)) # (RESET & !PHAN)               ;
0154 !        !W2    := AS # ( W0 & !W1 & !W2 ) # ( !W0 & !W1 & W2 ) # ( W1 & !W2 ) ;
0155 !        !W1    := AS # ( !W0 & W1 ) # ( W0 & !W1 )               ;
0156 !        !W0    := AS # ( W0 & W1 ) # ( W0 & W2  )                ;
0157 !
0158 !end pala1
```

B. PAL A DOCUMENT FILE

ABEL(tm) 3.00a - Document Generator 26-Jul-89 05:21 PM
Symbol list for Module pala1

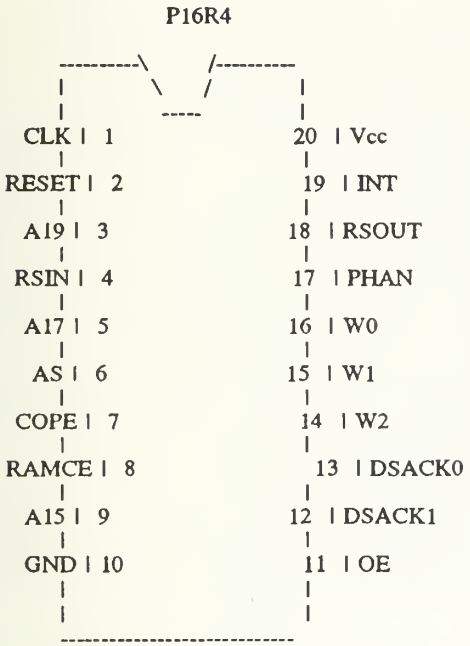
A15	Pin 9 pos, com
A17	Pin 5 pos, com
A19	Pin 3 pos, com
AS	Pin 6 pos, com
CK	(.C.)
CLK	Pin 1 pos, com
COPE	Pin 7 pos, com
DSACK0	Pin 13 neg, com
DSACK1	Pin 12 neg, com
INT	Pin 19 neg, com
OE	Pin 11 pos, com
PAL_A	device P16R4
PHAN	Pin 17 neg, reg, D
RAMCE	Pin 8 pos, com
RESET	Pin 2 pos, com
RSIN	Pin 4 pos, com
RSOUT	Pin 18 neg, com
S0	(0)
S1	(1)
S2	(2)
S3	(3)
S4	(4)
S5	(5)
S6	(6)
S7	(7)
W0	Pin 16 neg, reg, D
W1	Pin 15 neg, reg, D
W2	Pin 14 neg, reg, D
X	(.X.)
Z	(.Z.)
_PHAN_QN	Node 24 pos, com
_W0_QN	Node 23 pos, com
_W1_QN	Node 22 pos, com
_W2_QN	Node 21 pos, com
pala1	Module Name

Device PAL_A

- Reduced Equations:

```
RSOUT = !(A15 & A19);

INT = !(A17 & A19 & !RSIN);

DSACK0 = !(!AS & COPE & W0 & !W1 & !W2 # !AS & COPE & !RAMCE);

DSACK1 = !(A15 & !AS & !COPE # !AS & !RAMCE);

PHAN := !(!PHAN & RESET # !AS & !COPE);

W2 := !(W1 & !W2 # !W0 & !W1 & W2 # W0 & !W1 & !W2 # AS);

W1 := !(W0 & !W1 # !W0 & W1 # AS);

W0 := !(W0 & W2 # W0 & W1 # AS);
```

Device PAL_A


P16R4
CLK | 1
RESET | 2
A19 | 3
RSIN | 4
A17 | 5
AS | 6
COPE | 7
RAMCE | 8
A15 | 9
GND | 10
20 | Vcc
19 | INT
18 | RSOUT
17 | PHAN
16 | W0
15 | W1
14 | W2
13 | DSACK0
12 | DSACK1
11 | OE

ABEL(tm) 3.00a - Document Generator 26-Jul-89 05:21 PM
Fuse Map for Module pala1

Device PAL_A

```
      0         10        20        30
  0: --------- --------- --------- --
 32: ----X----X --X------- --------- --

256: --------- --------- --------- --
288: ----X----- --------- -------X- --

512: X--------- -X-------- --------- --
544: --------- -------X-- -X------- --

768: --------- ----X----- --X------- --
800: --------- ----X---X- --------- --
832: --------- ------X--- --------- --

1024: --------- ----X----X --------- --
1056: --------- -----X--X- --------- --
1088: --------- ------X--- --------- --

1280: --------- --------X- ---X------ --
1312: --------- -----X---X --X------- --
1344: --------- ----X----X ---X------ --
1376: --------- ------X--- --------- --

1536: --------- --------- --------- --
1568: --------- ----X--X-X X--X------ --
1600: --------- -------X-- X----X---- --

1792: --------- --------- --------- --
1824: --------- -------X-- -X------X- --
1856: --------- -------X-- -----X---- --
```

ABEL(tm) 3.00a - Document Generator 26-Jul-89 05:21 PM
for Module pala1

Device PAL_A

Device Type: P16R4 Terms Used: 22 out of 64

Pin #	Name	Terms Used	Max	Term Type	Pin Type
1	CLK	--	--	---	Clock
2	RESET	--	--	---	Input
3	A19	--	--	---	Input
4	RSIN	--	--	---	Input
5	A17	--	--	---	Input
6	AS	--	--	---	Input
7	COPE	--	--	---	Input
8	RAMCE	--	--	---	Input
9	A15	--	--	---	Input
10	GND	--	--	---	GND
11	OE	--	--	---	Enable
12	DSACK1	2	7	Normal	I/O
13	DSACK0	2	7	Normal	I/O
14	W2	4	8	Normal	Output
15	W1	3	8	Normal	Output
16	W0	3	8	Normal	Output
17	PHAN	2	8	Normal	Output
18	RSOUT	1	7	Normal	I/O
19	INT	1	7	Normal	I/O
20	Vcc	--	--	---	VCC
21	_W2_QN	--	--	---	Input (node)
22	_W1_QN	--	--	---	Input (node)
23	_W0_QN	--	--	---	Input (node)
24	_PHAN_QN	--	--	---	Input (node)

Device PAL_A

```
 1 [C--- -101 --0- ---- ---- ----] -> [---- ---- ---- HLLL ---- ----];
 2 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HHHH ---- ----];
 3 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HHHL ---- ----];
 4 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HHLH ---- ----];
 5 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HHLL ---- ----];
 6 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HLHH ---- ----];
 7 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HLHL ---- ----];
 8 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HLLH ---- ----];
 9 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HLLH ---- ----];
10 [C--- -001 --0- ---- ---- ----] -> [---- ---- ---- HLLH ---- ----];
11 [C--- -000 --0- ---- ---- ----] -> [---- ---- ---- HLLH ---- ----];
12 [C--- -101 --0- ---- ---- ----] -> [---- ---- ---- HLLL ---- ----];
13 [C--- -101 --1- ---- ---- ----] -> [---- ---- ---- HZZZ ---- ----];
14 [C--- -111 --0- ---- ---- ----] -> [---- ---- ---- HLLL ---- ----];
15 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- HHHH ---- ----];
16 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- HHHL ---- ----];
17 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- HHLH ---- ----];
18 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- HHLL ---- ----];
19 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- HLHH ---- ----];
20 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- HLHL ---- ----];
21 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- LLLH ---- ----];
22 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- LLLH ---- ----];
23 [C--- -011 --0- ---- ---- ----] -> [---- ---- ---- LLLH ---- ----];
24 [C--- -010 --0- ---- ---- ----] -> [---- ---- ---- LLLH ---- ----];
25 [C--- -111 --0- ---- ---- ----] -> [---- ---- ---- HLLL ---- ----];
26 [C--- -111 --1- ---- ---- ----] -> [---- ---- ---- HZZZ ---- ----];
27 [C0-- -01- --0- ---- 0--- ----] -> [---- ---- ---- ---- H--- ----];
28 [C0-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
29 [C0-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
30 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
31 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
32 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
33 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
34 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
35 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
36 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
37 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
38 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
39 [C0-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
40 [C0-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
41 [C0-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
42 [C0-- -00- --0- ---- 1--- ----] -> [---- ---- ---- ---- L--- ----];
43 [C0-- -01- --0- ---- 0--- ----] -> [---- ---- ---- ---- H--- ----];
44 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
45 [C1-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
46 [C1-- -00- --0- ---- 1--- ----] -> [---- ---- ---- ---- L--- ----];
47 [C1-- -00- --0- ---- 0--- ----] -> [---- ---- ---- ---- L--- ----];
48 [C1-- -01- --0- ---- 0--- ----] -> [---- ---- ---- ---- L--- ----];
49 [C1-- -01- --0- ---- 0--- ----] -> [---- ---- ---- ---- L--- ----];
50 [C0-- -01- --0- ---- 0--- ----] -> [---- ---- ---- ---- H--- ----];
51 [C0-- -01- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
```

Device PAL_A

```
 52 [C0-- -11- --0- ---- 0--- ----] -> [---- ---- ---- ---- H--- ----];
 53 [C0-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 54 [C0-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 55 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 56 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 57 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 58 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 59 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 60 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 61 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 62 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 63 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 64 [C0-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 65 [C0-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 66 [C0-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 67 [C0-- -10- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 68 [C0-- -11- --0- ---- 0--- ----] -> [---- ---- ---- ---- H--- ----];
 69 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 70 [C1-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 71 [C1-- -10- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 72 [C1-- -10- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 73 [C1-- -00- --0- ---- 0--- ----] -> [---- ---- ---- ---- L--- ----];
 74 [C1-- -10- --0- ---- 0--- ----] -> [---- ---- ---- ---- L--- ----];
 75 [C1-- -11- --0- ---- 0--- ----] -> [---- ---- ---- ---- L--- ----];
 76 [C0-- -11- --0- ---- 0--- ----] -> [---- ---- ---- ---- H--- ----];
 77 [C0-- -11- --0- ---- 1--- ----] -> [---- ---- ---- ---- H--- ----];
 78 [--0- ---- 0--- ---- ---- ----] -> [---- ---- ---- ---- -H-- ----];
 79 [--0- ---- 1--- ---- ---- ----] -> [---- ---- ---- ---- -H-- ----];
 80 [--1- ---- 0--- ---- ---- ----] -> [---- ---- ---- ---- -H-- ----];
 81 [--1- ---- 1--- ---- ---- ----] -> [---- ---- ---- ---- -L-- ----];
 82 [--00 0--- ---- ---- ---- ----] -> [---- ---- ---- ---- --H- ----];
 83 [--01 0--- ---- ---- ---- ----] -> [---- ---- ---- ---- --H- ----];
 84 [--00 1--- ---- ---- ---- ----] -> [---- ---- ---- ---- --H- ----];
 85 [--01 1--- ---- ---- ---- ----] -> [---- ---- ---- ---- --H- ----];
 86 [--10 0--- ---- ---- ---- ----] -> [---- ---- ---- ---- --H- ----];
 87 [--11 0--- ---- ---- ---- ----] -> [---- ---- ---- ---- --H- ----];
 88 [--10 1--- ---- ---- ---- ----] -> [---- ---- ---- ---- --L- ----];
 89 [--11 1--- ---- ---- ---- ----] -> [---- ---- ---- ---- --H- ----];
 90 [---- -010 ---- -000 ---- ----] -> [---- ---- ---- L--- ---- ----];
 91 [---- -01X ---- -001 ---- ----] -> [---- ---- ---- L--- ---- ----];
 92 [---- -11X ---- -XXX ---- ----] -> [---- ---- ---- H--- ---- ----];
 93 [---- -11X ---- -XXX ---- ----] -> [---- ---- ---- H--- ---- ----];
 94 [---- -X0X ---- -XXX ---- ----] -> [---- ---- ---- H--- ---- ----];
 95 [---- -000 0--- ---- ---- ----] -> [---- ---- ---L ---- ---- ----];
 96 [---- -001 0--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
 97 [---- -010 0--- ---- ---- ----] -> [---- ---- ---L ---- ---- ----];
 98 [---- -011 0--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
 99 [---- -000 1--- ---- ---- ----] -> [---- ---- ---L ---- ---- ----];
100 [---- -001 1--- ---- ---- ----] -> [---- ---- ---L ---- ---- ----];
101 [---- -010 1--- ---- ---- ----] -> [---- ---- ---L ---- ---- ----];
102 [---- -011 1--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
103 [---- -100 0--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
```

Device PAL_A

```
104 [---- -101 0--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
105 [---- -110 0--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
106 [---- -111 0--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
107 [---- -100 1--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
108 [---- -101 1--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
109 [---- -110 1--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
110 [---- -111 1--- ---- ---- ----] -> [---- ---- ---H ---- ---- ----];
```

end of module pala1

APPENDIX I : PAL B PROGRAMMING FILES

A. PAL B LISTING FILE

```
0001 !module  palb ;
0002 !flag  '-f'  ;
0003 !
0004 !"  MONOLITHIC MEMORIES INC.   PAL 16L8A-4 FAMILY/PINOUT CODE : 22/17
0005 !"  NATIONAL SEMICONDUCTOR     PAL 16L8A2  FAMILY/PINOUT CODE : 95/17
0006 !
0007 !        PAL_B DEVICE  'P16L8';
0008 !        RW,DS,S1,S0,A0,A1,P,A18,A17,GND          PIN 1,2,3,4,5,6,7,8,9,10 ;
0009 !        A15,ROMCE,RAM1W,RAM2W,RAM3W,RAMCE        PIN 11,12,13,14,15,16    ;
0010 !        RAM4W,COPE,RAMOE,VCC                     PIN 17,18,19,20          ;
0011 !        H,L,X   =  1,0,.X.                                                ;
0012 !
0013 !
0014 !        EQUATIONS
0015 !        !RAMCE = (!RW & P & !DS) # ( !A18 & !A17 & !P & !DS ) ;
0016 !
0017 !        !RAM1W = ( !A18 & !A17 & !RW & !A1 & !A0 & !DS  );
0018 !
0019 !        !RAM2W = ( !A18 & !A17 & !RW & !A1 & !S0 & !DS  )
0020 !                 # ( !A18 & !A17 & !RW & !A1 & A0 & !DS )
0021 !                 # ( !A18 & !A17 & !RW & !A1 & S1 & !DS );
0022 !
0023 !        !RAM3W = ( !A18 & !A17 & !RW & A1 & !A0 & !DS )
0024 !                 # ( !A18 & !A17 & !RW & !A1 & !S1 & !S0 & !DS )
0025 !                 # ( !A18 & !A17 & !RW & !A1 & S1 & S0 & !DS )
0026 !                 # ( !A18 & !A17 & !RW & !A1 & A0 & !S0 & !DS );
0027 !
0028 !        !RAM4W = ( !A18 & !A17 & !RW & S0 & S1 & A0 & !DS )
0029 !                 # ( !A18 & !A17 & !RW & !S1 & !S0 & !DS )
0030 !                 # ( !A18 & !A17 & !RW & A1 & A0 & !DS )
0031 !                 # ( !A18 & !A17 & !RW & A1 & S1 & !DS );
0032 !
0033 !        !ROMCE = (!A17 & RW & P & !DS ) # (A18 & RW & !P & !DS )     ;
0034 !
0035 !        !RAMOE = (!A18 & !A17 & RW & !P)                             ;
0036 !
0037 !        !COPE  = (!A18 & A17 & !A15)                                 ;
0038 !
0039 !test_vectors ( [A18,A17,A15,RW,P,A1,A0,S1,S0,DS] ->
0040 !                [RAMCE,RAM1W,RAM2W,RAM3W,RAM4W,ROMCE,RAMOE,COPE])
0041 !
0042 !"RAMCE
0043 !        [X,X,X,L,H,X,X,X,X,L] -> [L,X,X,X,X,X,X,X]                   ;
0044 !        [L,L,X,X,L,X,X,X,X,L] -> [L,X,X,X,X,X,X,X]                   ;
0045 !"RAM1W
0046 !        [L,L,X,L,X,L,L,X,X,L] -> [X,L,X,X,X,X,X,X]                   ;
0047 !"RAM2W
0048 !        [L,L,X,L,X,L,X,X,L,L] -> [X,X,L,X,X,X,X,X]                   ;
0049 !        [L,L,X,L,X,L,H,X,X,L] -> [X,X,L,X,X,X,X,X]                   ;
0050 !        [L,L,X,L,X,L,X,H,X,L] -> [X,X,L,X,X,X,X,X]                   ;
0051 !"RAM3W
0052 !        [L,L,X,L,X,H,L,X,X,L] -> [X,X,X,L,X,X,X,X]                   ;
0053 !        [L,L,X,L,X,L,X,L,L,L] -> [X,X,X,L,X,X,X,X]                   ;
0054 !        [L,L,X,L,X,L,X,H,H,L] -> [X,X,X,L,X,X,X,X]                   ;
0055 !        [L,L,X,L,X,L,H,X,L,L] -> [X,X,X,L,X,X,X,X]                   ;
0056 !"RAM4W
0057 !        [L,L,X,L,X,X,H,H,H,L] -> [X,X,X,X,L,X,X,X]                   ;
0058 !        [L,L,X,L,X,X,X,L,L,L] -> [X,X,X,X,L,X,X,X]                   ;
0059 !        [L,L,X,L,X,H,H,X,X,L] -> [X,X,X,X,L,X,X,X]                   ;
0060 !        [L,L,X,L,X,H,X,H,X,L] -> [X,X,X,X,L,X,X,X]                   ;
0061 !"ROMCE
0062 !        [X,L,X,H,H,X,X,X,X,L] -> [X,X,X,X,X,L,X,X]                   ;
```

```
0063 |        [H,X,X,H,L,X,X,X,X,L] -> [X,X,X,X,X,L,X,X]          ;
0064 |"RAMOE
0065 |        [L,L,X,H,L,X,X,X,X,X] -> [X,X,X,X,X,X,L,X]          ;
0066 |"COPE
0067 |        [L,H,L,X,X,X,X,X,X,X] -> [X,X,X,X,X,X,X,L]          ;
0068 |
0069 |
0070 |
0071 |end
```

B. PAL B DOCUMENT FILE

ABEL(tm) Version 2.00b - Document Generator 03-May-89 11:11 AM
Equations for Module palb

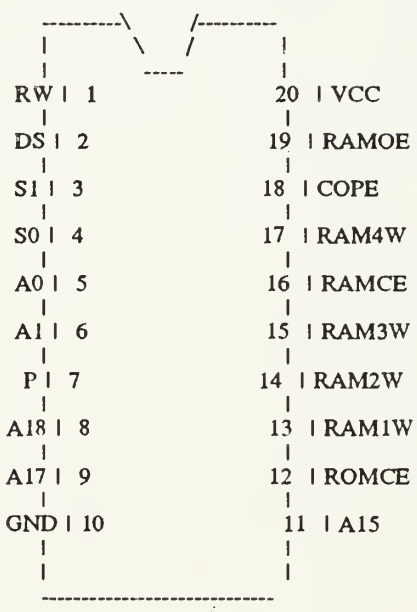
Device PAL_B

Reduced Equations:

```
RAMCE = !(!DS & P & !RW # !A17 & !A18 & !DS & !P);

RAM1W = !(!A0 & !A1 & !A17 & !A18 & !DS & !RW);

RAM2W = !(A0 & !A1 & !A17 & !A18 & !DS & !RW
       # !A1 & !A17 & !A18 & !DS & !RW & !S0
       # !A1 & !A17 & !A18 & !DS & !RW & S1);


RAM3W = !(!A0 & A1 & !A17 & !A18 & !DS & !RW
       # A0 & !A1 & !A17 & !A18 & !DS & !RW & !S0
       # !A1 & !A17 & !A18 & !DS & !RW & !S0 & !S1
       # !A1 & !A17 & !A18 & !DS & !RW & S0 & S1);


RAM4W = !(A0 & A1 & !A17 & !A18 & !DS & !RW
       # A1 & !A17 & !A18 & !DS & !RW & S1
       # !A17 & !A18 & !DS & !RW & !S0 & !S1
       # A0 & !A17 & !A18 & !DS & !RW & S0 & S1);


ROMCE = !(!A17 & !DS & P & RW # A18 & !DS & !P & RW);

RAMOE = !(!A17 & !A18 & !P & RW);

COPE = !(!A15 & A17 & !A18);
```

Device PAL_B

P16L8



ABEL(tm) Version 2.00b - Document Generator 03-May-89 11:11 AM
Fuse Map for Module palb

Device PAL_B

```
      0         10        20        30
   0: -------- --------- --------- --
  32: --X------ --------- -X---X---X --
  64: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
  96: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 128: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 160: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 192: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 224: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 256: -------- -------- --------- --
 288: --------- --------- -----X--X- -X
 320: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 352: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 384: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 416: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 448: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 480: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 512: --------- --------- ---------- --
 544: -X-X------ --X---X--- -----X---X --
 576: -X-XX----- ------X--- -----X---X --
 608: -X-X-X---X ---------- -----X---X --
 640: -X-XX---X- --X------- -----X---X --
 672: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 704: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 736: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 768: ---------- --------- ---------- --
 800: -X-X------ ---------- X--------- --
 832: -X-------- --------- -X---X---X --
 864: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 896: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 928: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 960: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
 992: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1024: ---------- ---------- ---------- --
1056: -X-X------ ---X--X--- -----X---X --
1088: -X-X-----X --X----X-- -----X---X --
1120: -X-X-X---X -------X-- -----X---X --
1152: -X-XX---X- -------X-- -----X---X --
1184: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1216: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1248: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1280: --------- --------- --------- --
1312: -X-X------ --X----X-- -----X---X --
1344: -X-X-----X -------X-- -----X---X --
1376: -X-XX----- -------X-- -----X---X --
1408: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1440: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1472: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1504: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1536: --------- --------- --------- --
1568: -X-X------ ---X---X-- -----X---X --
1600: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
```

ABEL(tm) Version 2.00b - Document Generator 03-May-89 11:11 AM
Fuse Map for Module palb

Device PAL_B

```
1632: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1664: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1696: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1728: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1760: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1792: ----------- ----------- ----------- --
1824: -XX-------- --------- X--------X --
1856: -XX-------- --------- -X--X------ --
1888: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1920: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1952: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
1984: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
2016: XXXXXXXXXXX XXXXXXXXXXX XXXXXXXXXXX XX
```

Device PAL_B

Device Type: P16L8 Terms Used: 26 out of 64

Pin #	Name	Terms Used	Max	Term Type	Pin Type
1	RW	--	--	---	Input
2	DS	--	--	---	Input
3	S1	--	--	---	Input
4	S0	--	--	---	Input
5	A0	--	--	---	Input
6	A1	--	--	---	Input
7	P	--	--	---	Input
8	A18	--	--	---	Input
9	A17	--	--	---	Input
10	GND	--	--	---	GND
11	A15	--	--	---	Input
12	ROMCE	2	7	Normal	Output
13	RAM1W	1	7	Normal	I/O
14	RAM2W	3	7	Normal	I/O
15	RAM3W	4	7	Normal	I/O
16	RAMCE	2	7	Normal	I/O
17	RAM4W	4	7	Normal	I/O
18	COPE	1	7	Normal	I/O
19	RAMOE	1	7	Normal	Output
20	VCC	--	--	---	VCC

end of module palb

LIST OF REFERENCES

1. Uzunsokakli, Y., "Design and Implementation of a Debugger for MC68020 Based Educational Computer Board", Master's Thesis, Naval Postgraduate School, Monterey, Ca , Dec 1989.

2. Motorola, MC68020 32-Bit Microprocessor User's Manual, Prentice-Hall, 1985.

3. Motorola, MC68881 Floating Point Coprocessor User's Manual, 1985.

4. Motorola, MC68000 Educational Computer Board User's Manual (MEX68KECB/D2), 1982.

5. FutureNet, ABEL 3.0 User's Guide, Jan 1988

6. G. J. Lipovski, 16- and 32-Bit Microcomputer Interfacing, Prentice-Hall, preprint.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Prof. Gerald J. Lipovski Department of Electrical and Computer Engineering The University of Texas Austin, Texas 78712	1
5.	Prof. Jon T. Butler, Code 62BU Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	Prof. C. Yang, Code 62YA Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
7.	Prof. Frederic Terman, Code 62TZ Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
8.	Hava Kuvvetleri Komutanligi Harekat Egitim Daire Baskanligi Bakanliklar - Ankara / TURKEY	1

9. 3. HIBM K.ligi
Fabrika Mudurlugu
Etimesgut - Ankara / TURKEY — 1

10. Yavuz TUGCU
Asagi Eglence Mercimek Sokak
41/20 Etlik - Ankara / TURKEY — 1

11. Mustafa Yavuz UZUNSOKAKLI
Sancaktepe Mahallesi
4/5 Sokak, No:1 Daire:4
Bagcilar - Bakirkoy
Istanbul / TURKEY — 1

12. M. Kadri HEKIMOGLU
SMC 2455 NPS
Monterey, California 93943 — 1

Thesis
T89
c.1
Tugcu
Design and implementation of an MC68020-based Educational Computer Board.